ADAPTIVE FUZZY LOGIC BASED FRAMEWORK FOR HANDLING IMPRECISION AND UNCERTAINTY IN SOFTWARE DEVELOPMENT EFFORT PREDICTION MODELS

BY

SYED ZEESHAN MUZAFFAR

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

INFORMATION & COMPUTER SCIENCE

JANUARY 2006

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **SYED ZEESHAN MUZAFFAR** under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfilment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE.**

Thesis Committee

M. Alshayeb
Dr. Mohammad Alshayeb (Chairman)

Muhammad Elahi
Dr. Moataz Ahmed (Co-Chairman)

Dr. Mamdouh Najjar (Member)

Dr. Jarallah Al-Ghamdi (Member)

Kanaan Faisal
Dr. Kanaan A. Faisal (Member)

Kanaan Faisal
Dr. Kanaan A. Faisal
Department Chairman

Dr. Mohammad A. Al-Ohali
Dean of Graduate Studies

١٤٢٧/٧/١٠
Date 11-7-2006

عمادة الدراسات العليا
DEANSHIP OF GRADUATE STUDIES

Dedicated to

My parents, sisters and brother

ACKNOWLEDGEMENT

In the name of Allah, Most Gracious, Most Merciful

All praise belongs to **Almighty Allah** (SWT), glorified is He and exalted. Who gave me the opportunity, courage, strength and persistence to carryout this work. And Who helped me in the most difficult of times. Peace and blessing of Allah be upon the last Prophet Muhammad (Peace Be upon Him).

Acknowledge is due to the King Fahd University of Petroleum & Minerals in extending a generous financial assistance to me and for providing a wonderful environment, academic and otherwise, which made my stay at KFUPM a memorable experience.

My deep and unrestrained appreciation goes to my thesis co-advisor Dr. Moataz Ahmed for his constant help, contribution and guidance and for his compassionate attitude. I cannot simply imagine how the things would have materialized without his guidance.

Very special thanks to my thesis advisor Dr. Mohammad Al-Shayeb for providing me the great moral support in carrying out this work.

I also wish to thank my thesis committee members, Dr. Mamdouh Najjar, Dr. Jarallah Al-Ghamdi and Dr. Kanaan A. Faisal for their help and support. Last but not least, I thank Dr. Muhammad Sarfraz for his help and encouragement.

I would like to thank my friend Quazi Abid-ur-Rahman for sparing his precious time for discussions that helped me a lot in conducting this work. I would also like to recognize the support of my other friends and colleagues with whom I spent excellent and fruitful company, Adnan yusuf, Syed Adnan Shahab, Akhtar Ghazi, Aiman Rasheed, Moin uddin, Saad Azhar, Mudassir Masood, Imran Naseem, Imran Azam, Khawar Saeed, Saad Mansoor, Abu Bakr, yusuf Sharif, Rehan Choudhry, Faisal Zaheer and Kashif Saeed. May Allah help them in all their future prospects and endeavours.

Finally I wish to express my gratitude to my parents and family members for being patient with me in all their hardships that they incurred in my long absence and offering words of encouragements to spur my spirit at the moments whenever needed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THESIS ABSTRACT

Name: Syed Zeeshan Muzaffar

Title: Adaptive Fuzzy Logic Based Framework for Handling Imprecision and Uncertainty in Software Development Effort Prediction Models

Major Field: Computer Science

Date of Degree: January 2006

The prediction of software development effort from early software estimates is a challenging task because not much of the information about the software is available at that time. Moreover, the information gathered for software attributes from various sources is subjective to imprecision and uncertainty. Imprecision arises when an expert defines some qualitative criteria to differentiate between two or more classes. Uncertainty arises due to the existence of more than one metric for a particular attribute. This uncertainty leads to the uncertainty in software effort prediction. This thesis presents a type-2 FLS-based effort prediction framework that is capable of handling imprecision and uncertainty. In addition, empirical studies regarding the impact of various parameters of FLS on effort prediction system are carried out. The thesis also identifies a set of attributes that can give indication of the credibility of existing size metrics; and uses these attributes to evaluate some existing software size metrics.

ملخص الرسالة

الاســـــــــم: سيد زيشان مظفر

عنوان الرسالة: التكيف المنطقي لحل الغموض وعدم الدقة في تطوير البرامج والتنبؤ وبذل الجهد.

التخصـــــص: علوم الحاسب الآلي والمعلومات

تاريخ التخـرج: ذو الحجة 1426

التنبؤ عن تطوير البرمجيات منذ البداية يمثل مهمه صعبه لانه ليس لدينا معلومات عن البرمجيات المتاحه في ذلك الوقت. التقديرات المبكره عاده ناقصه, وتتطور من خلال التكرار بمختلف انشطه البرامج العامة. علاوه علي ذلك ، فان المعلومات التي تم جمعها من مختلف البرامج غير معلومة المصادر وعادة تكون متصفة بالغموض وعدم اليقين. الغموض ينشا عندما يكون هناك خبير يعرف ببعض المعايير النوعيه للتفريق بين أي قسمين برمجيين او اكثر. وكذلك عدم التأكد يظهر لوجود اكثر من ميزة خاصه لكل قسم ، فمثلا الحجم بسبب تناقض اراء الخبراء حوله يؤدي هذا الشك الي الشك في الجهد المبذول على البرامج. هذه الاطروحه تمثل Type-2 FLS-Based. وهذا النوع من البرمجيات قادر على حل مشكلة الغموض وعدم الدقة. وبالاضافه الي ذلك ، فان الدراسات التجريبيه حول تاثير مختلف معالم نظام Type-2 FLS-Based انتهت. وهذه الاطروحة تعرف مجموعه من الصفات التي يمكن ان تعطي دلاله علي مصداقيه حجم المساحة البرمجية الحاليه. وتستخدم هذه الخصائص علي بعض البرامج الموجوده لتقييم حجم البرنامج.

CHAPTER 1

INTRODUCTION

Software development effort estimation deals with the prediction of the likely amount of cost, time and staffing level required to accomplish the required development task [37]. Typically, software development effort estimates are themselves based on the prediction of size of the future system, which is a difficult task in the sense that estimates obtained at the early stages of development life cycle are inaccurate because not much information of the future system is available at that time [19]. In order to understand the importance of reliable size prediction, consider a typical project prediction process which is depicted in Figure1-1. The figure depicts the initial requirements collection task as the first step in this process. The initial requirements collected are usually neither complete nor accurate and it takes several revisions to come to a final requirements specification. Therefore predicting size from such initial requirements is challenging and highly crucial because the rest of the project attributes prediction e.g., effort and cost, is highly dependent on this.

In addition, early estimates play a vital role in analyzing the cost-benefit and making decisions for contract bidding on a project [39]. Efficient strategies for planning and tracking the software projects are highly based on the careful and accurate cost and schedule estimation that helps in effectively controlling the expensive software development investment, which is of preponderant importance [46][68]. Underestimating the project effort results in under-staffing it which in turn causes staff burnout; also quite short a schedule results in loss of credibility as deadlines are missed. Meanwhile it is not a good practice to avoid this situation by generously incrementing estimates because over-estimating a project effort can be just about as bad for the organization. If one gives more resources to a project than it actually needs then it will cost more than it should, take longer to deliver and delay the use of blocked resources on other projects. According to Royce, a good and effective software cost estimate should fulfill the following properties [57]:

i. It is conceptualized and supported by the software project manager and the development team.
ii. It is acknowledged by all the stake holders as realizable.
iii. The underlying cost model is well-defined on a credible basis.
iv. It is based on a careful analysis of the pertinent historical project data (similar processes, similar technologies, similar environments, similar people and similar requirements).
v. It is defined in adequate detail such that its possible key risk areas are clearly understood and probability of success is objectively assessed.

The aforementioned needs suggest that the reliable and effective early effort prediction is a challenge in the field of software engineering. Because of the unavailability of effective and adequate information [26], software experts often rely on their past experience when predicting effort for software projects. The problem with such an approach is the non-repeatability and non-availability of highly expert estimators for every new project [22]. Moreover, we can get rough estimates from the past projects that share some similarity with the one under consideration, but each new project has its own requirements and attributes that contribute heavily in bringing-up uncertainty to the prediction process. Although researchers have put efforts to alleviate this situation by coming up with numerous models, frameworks and tools, these attempts were not successful enough. As a result, software industry suffers far more than it should. Accordingly, effort should be dedicated to improve the situation.

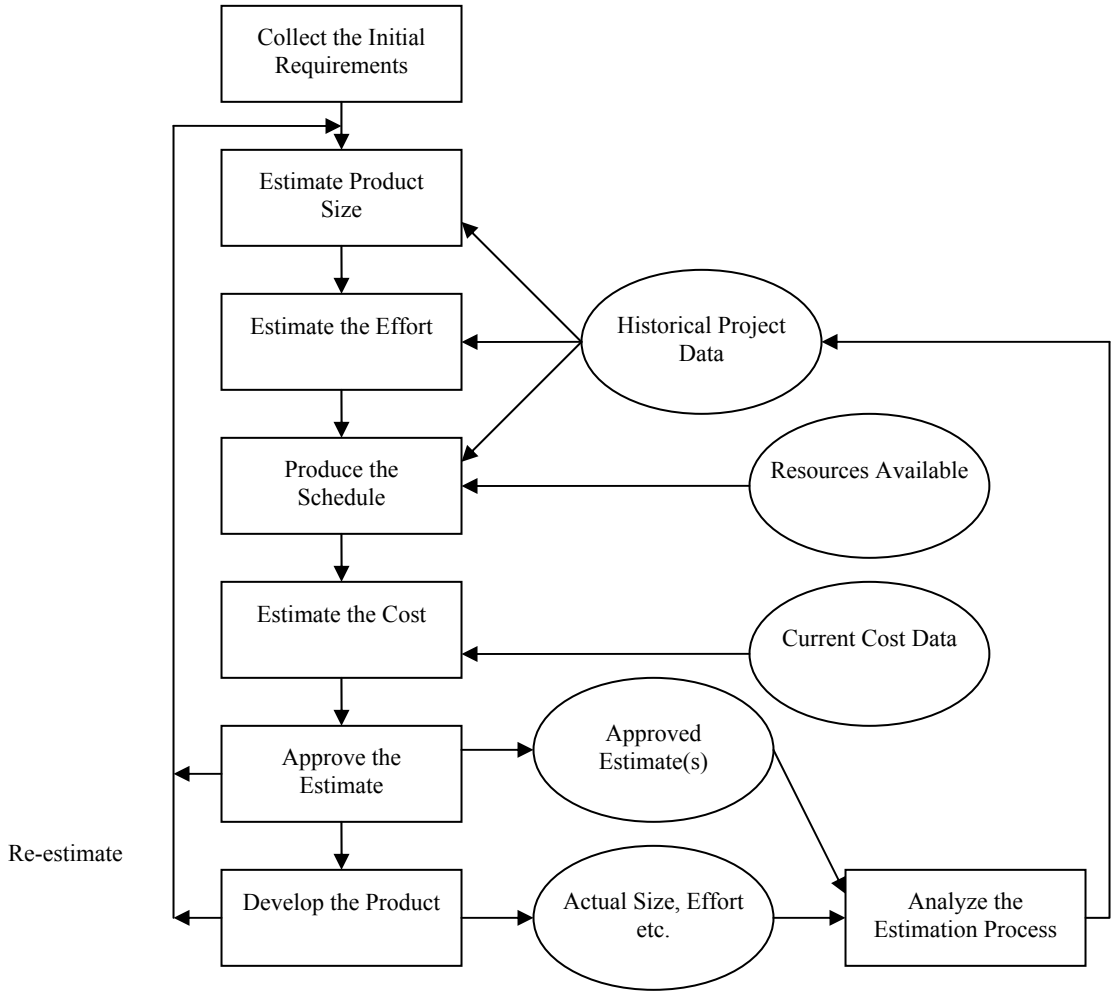

Figure 1-1: Project Estimation Process [35]

Existing software cost estimation techniques can be broadly classified as algorithmic and non-algorithmic models [62][22] . Algorithmic models are usually derived from the statistical analysis of the historical (past projects) data [64], for example Constructive Cost Model (COCOMO) [10], which is based on regression analysis and Software Life Cycle Management (SLIM) [55], which is based on Rayleigh distribution curves. Non-algorithmic techniques include Price-to-Win [10]; Parkinson [10]; Expert Judgment [4][10] and Machine Learning approaches [62][64]. Machine learning approaches include Fuzzy Systems, Regression Trees, Analogy, Rule Induction, Fuzzy Systems, Neural

Networks, Bayesian Networks and Evolutionary Computation. The last four of these approaches are categorized as Soft Computing techniques [62].

An excellent critical survey based on various soft computing based software cost estimation techniques can be found in [62]. The major results of the survey are:

i. Most of the state-of-the-art techniques make use of the historical data, which is seldom available. To tackle the problem of unavailability of sufficient amount of data some approaches have incorporated experts' knowledge.
ii. Adaptability is not the basic concern while developing those techniques. Therefore, they cannot cope with the rapidly changing development technologies and environment.
iii. Most of the approaches do not provide sufficient transparency for the accommodation of the expert knowledge in a reasonable manner.
iv. The experimental results of the soft computing based effort prediction approaches are not rigorous.
v. Various attempts have been made to incorporate fuzziness in various aspects of the COCOMO model. But these attempts lack the ability to integrate all of them in a single well-defined framework.
vi. Not a single model incorporates the adaptability and adequate transparency together.

In the light of their survey, Ahmed *et al.* [63] came up with a type-1 fuzzy logic based framework built on top of COCOMO effort prediction model. Their framework is categorized as:

i. A framework that incorporates the expert knowledge.

ii. A single well-defined framework that provides procedures to fuzzify various components of COCOMO model.

iii. Implementing training algorithms to incorporate adaptability.

Besides advantages, the major problem with their framework is its incapability to deal with uncertainty. The incapability comes from the fact that the basis for the framework is type-1 fuzzy logic system which itself is designed to tackle the imprecision in the fuzzy sets and not the uncertainty, due to noisy measurement or the different subjective opinions [49]. A detailed explanation of the drawbacks of their framework is provided in Chapter 4, whereas imprecision and uncertainty issues in the effort prediction are discussed in the following sections.

1.1. Imprecision and Uncertainty in Effort Prediction

Rahman [56] has outlined two categories of information, as shown in Figure 1-2, used to estimate an external attribute (e.g., cost, effort, reliability etc) using some internal attributes (e.g., size, number of faults etc.) given an underlying model: Numerical Information and Linguistic Information. Each category is further classified as Assessment and Relationship. Numerical assessment comes from the corresponding metrics (e.g., statistical regression based approaches like COCOMO) where as linguistic assessment, coming from the experts' judgment when they use words to provide valuable information (e.g., high size, medium effort, low complexity etc).

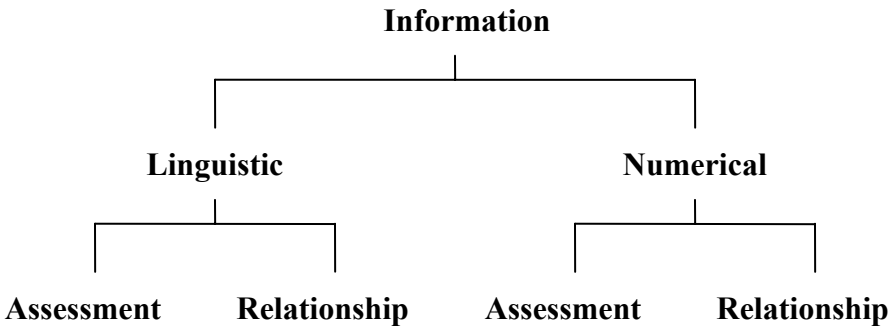

Figure 1-2: Sources of information in estimation models

A numerical relationship is a quantitative relationship between internal and external attributes over a range of values (e.g., size values of 10 to 20 KDSI require an effort of 150 to 180 person months). On the other hand, a linguistic relationship is an expert-provided relationship between internal and external attributes using sentences that make use of words (e.g., low size may require medium effort).

The success of any prediction model (e.g., effort or defects prediction models) that makes use of one or more sources of information is based on the availability of good historical data and experts' opinion. A prediction model that incorporates historical data and experts' opinion has two inherent problems: imprecision and uncertainty. Imprecision arise when an expert defines some qualitative criteria to differentiate between two or more classes. This is because expert is a human being whose knowledge is imprecise in nature due to the representation of knowledge in words. Moreover, the criteria defined are based on his past experience and again it may not be 100% precise. Therefore historical data is used to make the boundaries between the classes precise.

Uncertainty has two sources [56]: measurement and linguistic uncertainties. Measurement uncertainty comes into play when two or more different metrics measure the same quality attribute (e.g., size or cohesion etc.). The uncertainty arises due to the difference in the underlying models that are used to measure the quality attribute. This difference produces different measurements, which in turn causes uncertainty to the software engineer as which measurement should be used to capture the desired assessment effectively. The consequence of this is that the uncertainty in the internal attributes gives rise to uncertainty in the corresponding external attributes. Adam *et al* [1] present an example of such inconsistencies in metrics' measurements when trying to measure cohesions. One thing that must be made clear at this point is that any individual metric is certain in itself because given the same dataset, it will always produce the same outcome; exceptions apply only to metrics that are non-deterministic.

Moreover, other factors that contribute to the measurement uncertainty are laziness and ignorance and to some extent the accuracy of the underlying model. Laziness comes into play when software engineer does not consider all the internal attributes that affect a particular external attribute, whereas ignorance means the lack of knowledge to consider all the internal attributes that can affect a particular external attribute. Chapter 2 elaborates more on the different sources of uncertainty.

Linguistic uncertainty is concerned with the different understanding of the same term by different people (experts) e.g., the term LOW might have different meaning to different experts, therefore experts may have slightly different opinion when judging artifacts (e.g.,

software components). For example, considering the size of a software component; one expert may rate this as of large size, while other may rate the same component as of moderate size. On the other hand, experts may differ when assigning different size intervals to LOW. One may say 0-10 is LOW, while other may interpret 0-12 as LOW and so on. Finally, experts generally differ in their judgments on the impact of certain internal attributes on the corresponding external attributes. For example, one expert may say "large software size requires a big amount of effort", while another may assert "large software size requires a moderate amount of effort".

1.2. Problem Statement

The importance of the consequences emerged due to the presence of imprecision and uncertainty have been identified in the above discussion. A detailed literature review reveals that researchers have put their effort to provide reasonable solutions to the problem of cost/effort prediction using algorithmic and non-algorithmic approaches [62]. But to date, no software development effort prediction system is capable of handling imprecision and uncertainty within a single framework. This work will discuss various possible sources of uncertainty in building effort prediction system using historical data and expert opinion. Our aim is to employ a Fuzzy Logic System (FLS) for effort prediction with the existence of imprecision and uncertainty. Our literature survey, discussed in Chapter 4, shows that the work of Ahmed *et al* [2] is the only framework that provides a complete fuzzy logic based framework for effort prediction. Accordingly, this proposed research suggests building on top of their framework in trying to handle uncertainty. Moreover, Ahmed *et al[2]* and Saliu [63] have outlined some directions for

future work, e.g., investigating the impact of the different parameters of the FLS on the effort prediction accuracy. These parameters include the membership functions, the training algorithms and the defuzzification methods. This work carries out these investigations and compares the results with their results. This work also investigates the possibility of combining both the cost drivers' information together with the mode and size in one-stage estimation rather than two stages as proposed by Ahmed *et al*.

The effort prediction framework presented in this thesis and the other prediction systems [26][38][83] presented in the literature are usually based on reliable prediction of software size. Therefore, a number of different size metrics are proposed by various researchers. Due to this importance of software size metrics, a detailed literature review of the existing software size metrics is carried out in this thesis. This literature review reveals that software engineering community has been suffering from the lack of a detailed critical survey of proposed size metrics. Besides this there is no well defined criterion to evaluate size metrics for their efficacy. Thus we will look into this problem and come up with a set of attributes that can help evaluating software size metrics. We will then evaluate some of the prominent size metrics on the basis of identified attributes set.

1.3. Main Contributions

The main contributions of this work are as follows.

i. Identification of various attributes that portray a picture about the credibility of software size metrics as good predictors for effort, and evaluation of existing size metrics on the basis of these attributes.

ii. Identification of four classes of FLS-based effort prediction systems. The classification is based on the nature of inputs e.g., precise vs. imprecise; and on whether uncertainty is considered in the system.

iii. Development of a framework for handling imprecision and uncertainty due to size measures, experts' opinion, and laziness/ignorance.

iv. Study of the impact of different architectures on the performance of FLS-based effort prediction framework. Here architecture stands for the combination of various building blocks of the framework i.e., how different inputs and the intermediate products are combined to produce final external attribute. Framework, on the other hand, portrays a bigger picture that includes definition of various inputs, fuzzy sets, algorithms and architecture.

v. Study of the impact of various parameters of FLS on the accuracy of the resultant effort prediction systems.

vi. Study of the impact of training approaches e.g., steepest descent and heuristic based, on the effort prediction accuracy of type-1 FLS systems.

1.4. Organization of the Thesis

The rest of the thesis is organized as follows. Chapter 2 discusses the preliminaries required to understand the work presented in this thesis. Chapter 3 presents a critical survey of various software size metrics on the basis of some identified attributes. Chapter 4 provides a brief literature review of the existing effort prediction techniques. Chapter 5 presents new framework for handling uncertainty due to noise and laziness/ignorance. Chapter 6 presents various training algorithms, parameters and architectures and their impacts on the effort prediction accuracy. Chapter 7 discusses experimental setups and results. At the end, Chapter 8 concludes the thesis pointing out the contributions and providing directions for the future work.

CHAPTER 2

BACKGROUND

Due to the importance of COCOMO, Fuzzy Logic System, Imprecision and Uncertainty in our research we provide a brief overview on them in the following sub sections.

2.1. The Constructive Cost Model (COCOMO)

The COCOMO is a regression based software cost estimation model that was developed by Barry Bohem [10] in 1981. It is thought to be the most plausible [51], best known [38] and the most cited [26] of all traditional cost prediction models. COCOMO can be used to calculate the amount of effort and the time schedule for software projects. There have been a few different versions of COCOMO; among these versions, COCOMO 81[10] and COCOMO II [9] are used in this thesis.

2.1.1. COCOMO 81

COCOMO 81 describes three different models that can be used throughout a project's life cycle [10]

i. Basic Model – this model is applied early in a project development. It provides a rough estimate early on that should be refined later on with one of the other models.

ii. Intermediate Model – this model is used when one has more detailed requirements for a project.

iii. Advanced Model – this model can be used to refine one's estimates when one has complete design for a project; it incorporates additional cost drivers.

Each of these models distinguishes between three basic development modes: *Organic*, *Embedded* and *Semi-detached*. The selection of a particular mode depends on work environment, and other constraints on the project itself. Organic mode is used for relativity small software teams developing software of typically low complexity in a highly familiar in-house environment. Embedded mode software projects operate within tight constraints e.g., real-time systems, whereas Semi-detached mode is somewhere intermediate stage between organic and embedded modes [52].

The intermediate COCOMO model is the most widely used version. It has estimation accuracy that is considerably greater than the basic model and comparable to that obtained using the advanced model [27]. The intermediate COCOMO model estimates effort in Person Months and is given using the formula:

$$PM = A \times [Size]^{B} \times EAF \quad (2\text{-}1)$$

- PM is the effort in person-months
- EAF is the effort adjustment factor
- Size is thousands of delivered source instructions
- B is the project development mode and also known as a scaling factor
- A is constant

The constants and scale factors for different modes are specified by Boehm as shown in Table 2-1[10].

Table 2-1: COCOMO mode coefficient and scale factor values

Mode	A	B
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.2

The EAF is used to adjust one's estimate based on various attributes of the development environment. It is not used in the basic model, where it is set to 1. The intermediate model defines 15 different cost drivers that can be used to calculate EAF. They are grouped into 4 different categories; *product*, *computer*, *personal* and *project* attributes, see Table 2-2. Cost drivers have up to six levels of rating: *Very Low, Low, Nominal, High, Very High,* and *Extra High*. These ratings are based on a statistical analysis of

historical data collected from 83 past projects [10]. Each rating has a real number (effort multiplier), based upon the factor and the degree to which the factor can influence productivity. To calculate the EAF from the cost drivers, one simply chooses a rating for each cost driver and multiplies them all together.

Table 2-2: List of 15 costs drivers and their ratings for COCOMO 81 [10][79]

Category	**Cost Driver**	**Very Low**	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
Product Attributes	RELY: Software Reliability	0.75	0.88	1.00	1.15	1.40	-
	DATA: Database Size	-	0.94	1.00	1.08	1.16	-
	CPLX: Product Complexity	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes	TIME: Execution Time Cons.	-	-	1.00	1.11	1.30	1.66
	STOR: Main Storage Constraint	-	-	1.00	1.06	1.21	1.56
	VIRT: Virtual Machine Volatility	-	0.87	1.00	1.15	1.30	-
	TURN: Turnaround Time	-	0.87	1.00	1.07	1.15	-
Personnel Attributes	ACAP: Analyst Capability	1.46	1.19	1.00	0.96	0.71	-
	AEXP: Applications Experience	1.29	1.13	1.00	0.91	0.82	-
	PCAP: Programmer Capability	1.42	1.17	1.00	0.86	0.70	-
	VEXP: Virtual Machine Experience	1.21	1.10	1.00	0.90	-	-
	LEXP Language Experience	1.14	1.07	1.00	0.95	-	-
Project Attributes	MODP: Modern Programming Practices	1.24	1.10	1.00	0.91	0.82	-
	TOOL: Use of Software Tools	1.24	1.10	1.00	0.91	0.83	-
	SCED: Development Schedule	1.23	1.08	1.00	1.04	1.10	-

The advanced model of COCOMO 81 goes one step further than the intermediate model in that it rates cost drivers differently depending on the current phase of the project development.

2.1.2. COCOMO II

One of the problems with using COCOMO 81 today is that it does not match the development environment of the late 1990's and 2000's. It was created in a time when batch jobs were the norm; programs used to run on mainframes and compile times were measured in hours instead of seconds [14]. It is outdated for use in today's development environment (rapid application development, 4th generation languages etc); therefore in 1997 COCOMO II was published and was supposed to solve most of these problems. This is also the reason as to why we opt for developing a framework by utilizing COCOMO II attributes. The main objectives of COCOMO II were set out when it was first published. They are [9]:

i. To develop software cost and schedule estimation model tuned to the life cycle practices of the 1990's and 2000's.
ii. To develop software cost database and tool support capabilities for continuous model improvement.
iii. To provide a quantitative analytic framework, and set of tools and techniques for evaluating the effects of software technology improvements on software life cycle costs and schedules.

COCOMO II estimates are obtained in pretty much the same way as COCOMO 81. The main differences are in the number and type of cost drivers and the calculation of

equation variables rather than the use of constants (for a detailed look at the specific differences between COCOMO 81 and COCOMO II see [1]). The equations still use lines of code as their main metric, one can however also use function points and object points for estimation.

COCOMO II has three models also, but they are different from those of COCOMO 81. They are [9]:

i. Application Composition Model – this is used for projects built using rapid application development tools. Normally, with this model, one uses object points as size estimates.

ii. Early Design Model – This model can provide one with estimates early in a projects design before the entire architecture has been decided on. Normally one would use function points as a size estimate with this model. It involves exploration of alternative software/system architectures and concepts of operation. At this stage, not enough is generally known to support fine-grain cost estimation.

iii. Post-Architecture Model – The most detailed on the three, used after the overall architecture for the project has been designed. One could use function points or LOC as size estimates with this model. It involves the actual development and maintenance of a software product.

COCOMO II describes 17 cost drivers that are used in the Post-Architecture model. They are used in the same way as in COCOMO 81 to calculate the EAF. The cost drivers, themselves, are different form those in COCOMO 81 though; they are better

suited for the software development environment of 1990's and 2000's. They are grouped together as shown in Table 2-3. More specific details on all of the cost drivers can be found in [9]. The cost drivers for COCOMO II are again rated on a scale from Very Low to Extra High in the same way as in COCOMO 81.

Table 2-3: List of Cost Drivers for COCOMO II [9]

Category	**Cost Driver**
Product Factors	RELY: Required Software Reliability
	DATA: Data Base Size
	CPLX: Product Complexity
	RUSE: Required Reusability
	DOCU: Documentation match to life-cycle needs
Platform Factors	TIME: Execution Time Constraint
	STOR: Main Storage Constraint
	PVOL: Platform Volatility
Personnel Factors	ACAP: Analyst Capability
	PCAP: Programmer Capability
	AEXP: Applications Experience
	PEXP: Platform Experience
	LTEX: Language and Tool Experience
	PCON: Personnel Continuity
Project Factors	TOOL: Use of Software Tools
	SITE: Multi-site Development
	SCED: Required Development Schedule

2.2. Fuzzy Logic

Fuzzy logic was first proposed and coined by Lotfi A. Zadeh, Professor of Systems Theory at the University of California, Berkeley, USA, in a publication in 1965[80]. Fuzzy Logic was the term given to a system of mathematics developed to model the human brain's curious way of processing and selecting words. The main motivation behind fuzzy logic was the existence of imprecision in the measurement process. In Zadeh's own words: "**As complexity rises, precise statements lose meaning and meaningful statements lose precision"** [80]. The next subsection is dedicated to define fuzziness more precisely and subsequent subsections discuss about fuzzy set theory and basic fuzzy logic system in brief. Some of the definitions and discussions here are due to Wang's book on adaptive fuzzy logic [76], Mendel's book on uncertain rule based FLS [49], Master's Thesis by Saliu [63] and Master's Thesis by Rahman [56].

2.2.1. Imprecision

Imprecision is associated with a lack of precise knowledge. We sometimes have measurements that are inaccurate, inexact, or associated with low confidence. Imprecision is the ambiguity, vagueness or subjectivity in natural language. It is the ambiguity found in the definition of a concept or the meaning of terms such as "tall tower" or "low blood pressure". It is also the ambiguity in human thinking, that is, perceptions and interpretations. Examples of statements that are fuzzy in nature are "Vitamin B quantity is very low" and "Mehmood is rather heavy compared to Akhtar".

2.2.2. Fuzzy Sets and Linguistic Variables

L. Zadeh defines fuzzy logic in the foreword of Wang's book [76]: **"In a broader and much significant sense, fuzzy logic is coextensive with the theory of fuzzy sets, that is, classes of objects in which the transition from membership to non-membership is gradual rather than abrupt"**. So, before defining a fuzzy logic system, fuzzy sets and linguistic variables should be explored first.

Linguistic Variables, Linguistic Values, Linguistic Terms: In fuzzy logic, linguistic variables accepts linguistic values which are words (linguistic terms) with associated degrees of membership in the set. Therefore, instead of considering length as a numerical variable that assumes a numerical value of 1.72 meters, it is treated as a linguistic variable that may assume, for example, linguistic values of *"high"* with a degree of membership of 0.92, *"short"* with a degree of 0.06, or *"medium"* with a degree of 0.7. This concept was introduced by Zadeh to provide a means of approximate characterization of phenomena that are too complex or too ill defined to be amenable to description in conventional quantitative terms [80].

Linguistic variables accept values defined in their term set - their set of linguistic terms. Linguistic terms are subjective categories for the linguistic variable. For example, for linguistic variable *age*, the term set *T(age)* may be defined as follows:

T(age) = { "young", "not young", "not so young", "very young", ..., "middle aged", "not middle aged", ..., "old", "not old", "very old", "more or less old", "quite old", ..., "not very young and not very old", ... }

Fuzzy Sets and Membership Functions: Each linguistic term is associated with a fuzzy set, each of which has a defined membership function (MF). Formally, a fuzzy set *A* in *U* is expressed as a set of ordered pairs

$$A = \{(x, \mu_A(x)) \mid x \, in \, U\}$$

In the above definition $\mu_A(x)$ is the membership function, which provides the degree of membership of *x*. This indicates the degree to which *x* belongs in set *A*, where *U* is the universe of discourse. Let's illustrate these concepts using an example. Consider the LOC is a metric to measure the size of a program in terms of number of lines of code. Figure 2-1 illustrates a linguistic variable LOC with three associated linguistic terms namely "low", medium" and "high". Each of these linguistic terms is associated with a fuzzy set defined by a corresponding membership function.

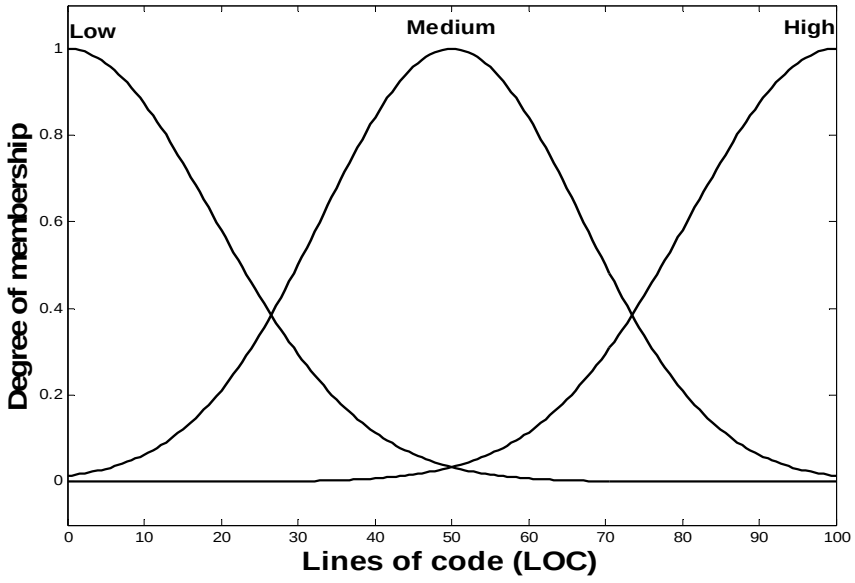

Figure 2-1: Membership functions for LOC

There are many types of membership functions. Some of the more common ones are triangular MFs (such as the functions in Figure 2-1), trapezoidal MFs, Gaussian MFs, and generalized bell MFs [53].

2.2.3. Fuzzy Logic System

Fuzzy logic system is a system which has a direct relationship with fuzzy concepts (such as fuzzy sets, linguistic variables and so on) and fuzzy logic. The most popular fuzzy logic systems in the literature can be classified into three types: pure fuzzy logic systems, Takagi and Sugeno's fuzzy system, and fuzzy logic system with fuzzifier and defuzzifier [76]. Since most of the engineering applications produce crisp data as input and expects crisp data as output, the last type is the most widely used one [76]. Figure 2-2 shows the basic configuration of a fuzzy logic system with fuzzifier and defuzzifier.

This type of fuzzy logic system was first proposed by Mamdani [47]. It has been successfully applied to a variety of industrial processes and consumer products [76]. The main fours components' functions are as follows.

Fuzzifier: It converts a crisp input to a fuzzy set.

Fuzzy Rule Base: Fuzzy logic systems use fuzzy IF-THEN rules. A fuzzy IF THEN rule is of the form "*IF* X_1 = A_1 *and* X_2 = A_2 ... *and* X_n = A_n *THEN* Y = B" where X_i and Y are linguistic variables and A_i and B are linguistic terms. The IF part is the antecedent or premise, while the THEN part is the consequence or conclusion. An example of a fuzzy

IF-THEN rule is "*IF Size = Low THEN Effort =High*". In a fuzzy logic system, the collection of fuzzy IF-THEN rules is stored in the fuzzy rule base, which is known as the inference engine.

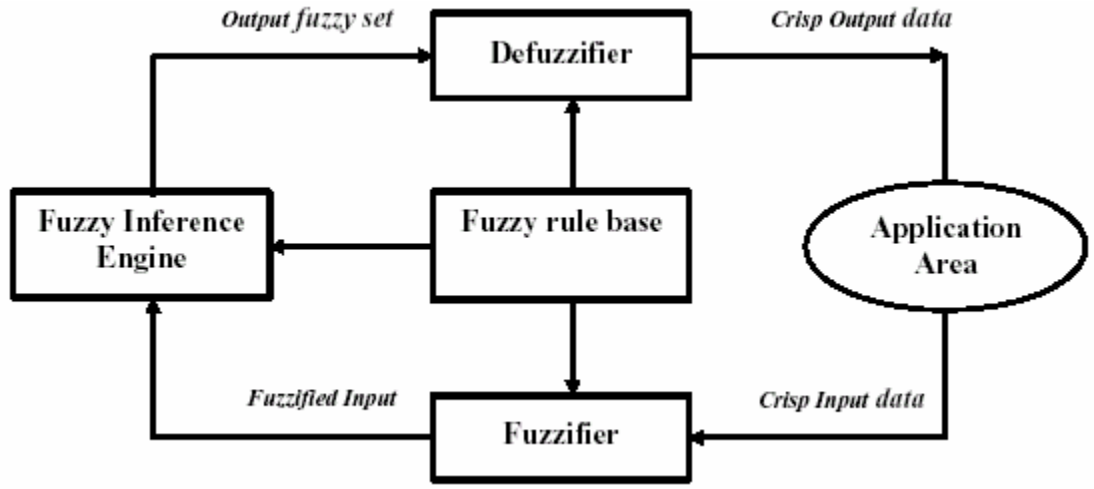

Figure 2-2: Fuzzy logic system with fuzzifier and defuzzifier

Fuzzy Inference Engine: Once all crisp input values are fuzzified into their respective linguistic values, the inference engine accesses the fuzzy rule base to derive linguistic values for the intermediate and the output linguistic variables. The inference engine performs two main operations: aggregation and composition. Aggregation is the process of computing for the values of the IF (antecedent) part of the rules while composition is the process of computing for the values of the THEN (conclusion) part of the rules.

Defuzzifier: It converts fuzzy output into crisp output.

The details of the above four components can be found in Wang's book [76].

2.2.4. Adaptive Fuzzy Logic

The definition of adaptive fuzzy system given by Wang in his book [76] is appropriate one: "**An adaptive fuzzy system is defined as a fuzzy logic system equipped with a training algorithm, where the fuzzy logic system is constructed from a set of fuzzy IF-THEN rules using fuzzy logic principles, and the training algorithms adjust the parameters of the fuzzy logic system based on numerical information"**. Here parameters e.g., position, mean or standard deviation etc., are the necessary values to construct the membership functions. Membership functions are adjusted by a set of input output pairs. So, adaptive fuzzy logic is a nice way of combining linguistic and numerical information, which can be done in two ways [76]:

i. Use of linguistic information (experts' knowledge) to develop an initial fuzzy logic system, and then adjust the parameters of the initial fuzzy logic system by using on numerical information.
ii. Use of numerical information and linguistic information to develop two separate fuzzy logic systems, and then final fuzzy logic system is obtained by averaging them.

In the first case components of the FLS are set based on the expert's opinion. These components include number of rules, shape and position of the membership functions and the shape and position of the consequents etc. Historical data is then used to further tune the parameters of the fuzzy logic system. The second way of coming up with adaptive FLS is straight forward in the sense that once the parameters of the individual systems are available, one can average these parameters to produce a final FLS.

It is clear from the above discussion that in a fuzzy logic system, the experts are the valuable source of information for establishing rules. The rules are then treated using the historical data. But the problem with this classical fuzzy logic (also known as type-1 Fuzzy Logic after the advent of type-2 Fuzzy logic discussed below) system is that it only handles the imprecision but not the uncertainty as described in Chapter 1. In order to handle imprecision and uncertainty simultaneously, one needs to establish the prediction system using the type-2 fuzzy logic, whish is expected to be capable of handling types of uncertainties discussed in Chapter 1 [49].

2.3. Uncertainty

Uncertainty is a very important aspect of human life. By the dictionary definition, it means, "Not knowing with certainty, doubtful; not definitely known; such as cannot be definitely forecast; subject to chance; not to be depended on; changeable" [59]. The uncertainty occurs mainly due to three reasons [59]:

i. Volume of work: It requires too much effort to list all the antecedents and consequences in the problem domain.
ii. Theoretical Ignorance: We usually do not have the sufficient knowledge of the domain to list every consideration.
iii. Practical Ignorance: It may be possible that all the tests are not available, or we do not want to run all the tests.

2.3.1. Uncertainty in Fuzzy Logic Systems

Mendel [49] has noted that uncertainty exists while building and using typical fuzzy logic systems. He has described four sources of uncertainty. Those are summarized here.

i. *Uncertainty about the meanings of the words that are used in a rule.* This is the uncertainty associated with the antecedent and/or consequent membership functions because membership functions represent words in a FLS.

ii. *Uncertainty about the consequent that is used in a rule.* A rule defines the impact of the antecedents on the consequent. Experts may vary in their judgment to decide this nature of impact.

iii. *Uncertainty about the measurements that activate the FLS.* This is the uncertainty with the input values or measurements that activates the FLS systems. These measurements may be precise or imprecise.

iv. *Uncertainty about the data that are used to tune the parameters of a FLS.* This is the uncertainty with the historical data, which is used to train the FLS as opposed to that of iii) which are used to activate the FLS.

2.3.2. Uncertainty and Type-2 Fuzzy Sets

Mendel has proposed using type-2 fuzzy sets and type-2 fuzzy logic systems to deal with the four types of uncertainties discussed in the previous section. Type-2 fuzzy sets were first proposed by Zadeh [81] in 1975. But the characterization of type-2 fuzzy sets was first done by Mendel and Liang in 1999 [48]. They characterized type-2 fuzzy sets using the concept of footprint of uncertainty and upper and lower membership functions. The type-2 fuzzy set is three dimensional whereas type-1 is two dimensional. The extra

dimension allows handling the possible uncertainties. This section presents type-2 fuzzy sets and how they can help to model uncertainty. Type-2 fuzzy sets help us to deal with the first source of uncertainty i.e. uncertainty about the meaning of the words. Type-1 fuzzy sets can not deal with this type of uncertainty because the degree of membership is considered as certain in type-1 fuzzy sets.

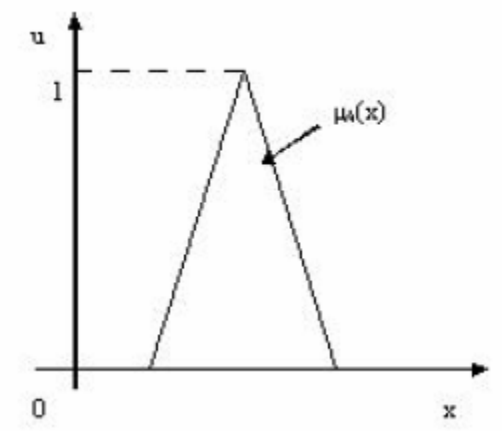

Figure 2-3: A type-1 triangular membership function

Let's imagine blurring the type-1 membership function depicted in Figure 2-3 by shifting the points on the triangle either to left or to right and not necessarily by the same amounts, as in Figure 2-4. This results in a shape such that at any specific value of x, say x′, there is no longer a single value for the membership function on y axis; instead the membership function takes a range of values wherever the vertical line intersects the blur.

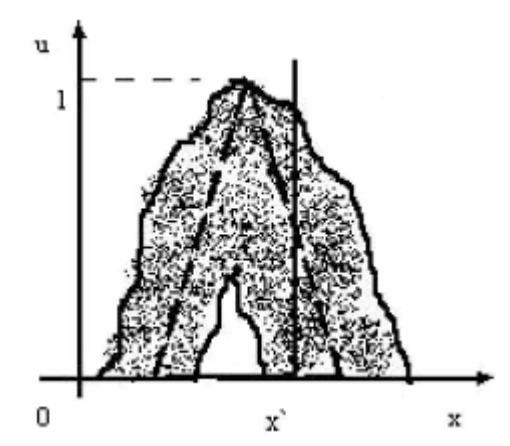

Figure 2-4: Blurred triangular membership function

It is not necessary to weight all the y values uniformly; hence, an amplitude distribution to all those values can be assigned. Performing this for all $x \in X$, a three-dimensional membership function, which is a type-2 membership function that characterizes a type-2 fuzzy set, is created. Type-2 membership functions have same constraint of type-1 membership functions. The degree of membership along the second dimension lies in the interval [0, 1]. The amplitude distribution i.e. the values along the 3rd dimension, also lies within the interval [0, 1]. So, if the blur disappears, then a type-2 membership function will reduce to a type-1 membership function.

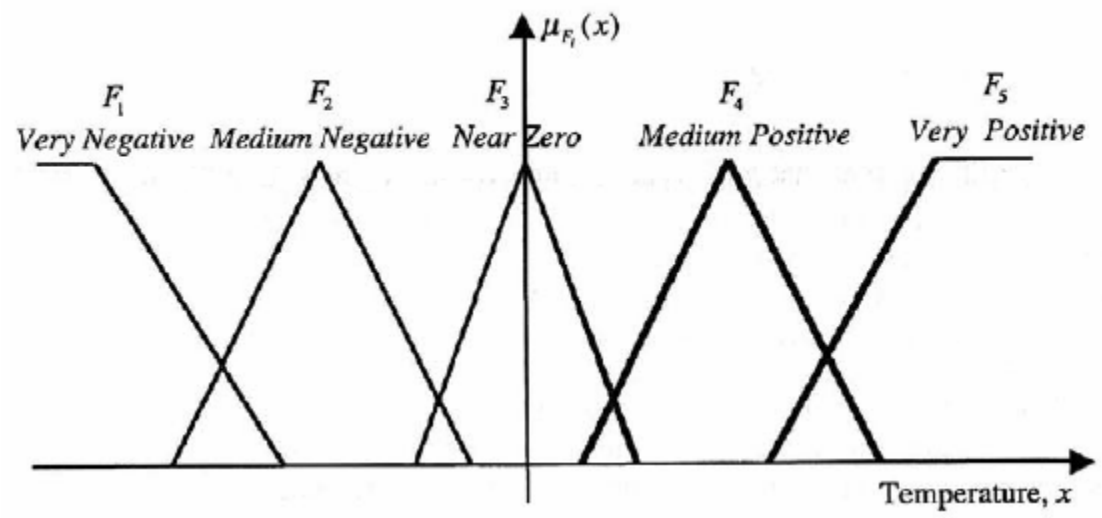

Figure 2-5: Type-1 Fuzzy sets

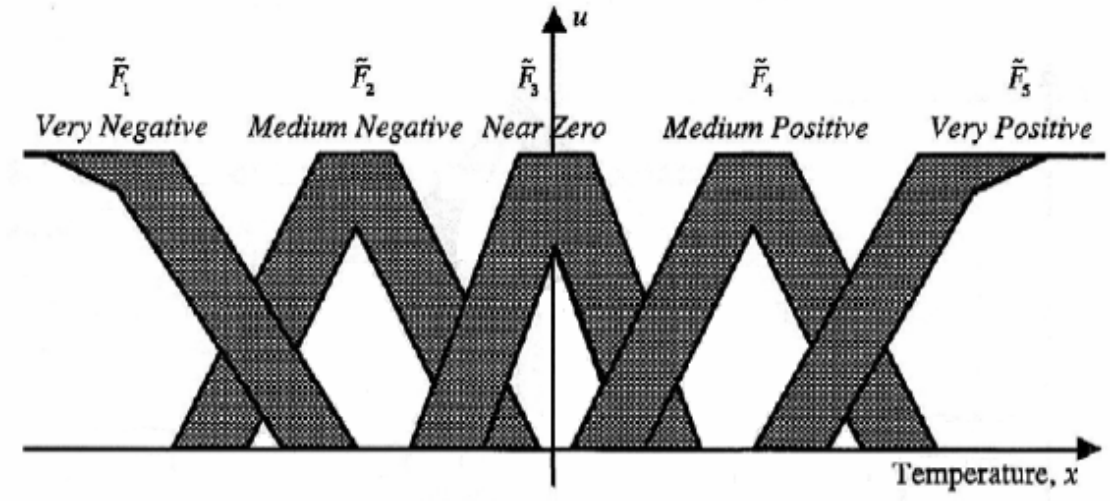

Figure 2-6: FOUs for membership functions of Figure 2-5

Mendel describes the blurred area as footprint of uncertainty (FOU). Figure 2-5 shows some triangular membership functions and Figure 2-6 shows FOUs for those membership functions [49]. The example shown in Figure 2-6 depicts a case where the FOU is uniformly shaded. It means that at each point in the FOU, the membership degree is one. This type of membership functions is known as interval type-2 membership function.

Imposing this constraint enables to easily manipulate the mathematics of the corresponding fuzzy logic system, and accordingly build the inference engine. In order to be able to re-use the mathematical results presented by Mendel, we have used Gaussian membership functions in our experiments to build the fuzzy logic systems. Figure 2-7 depicts an example of a Gaussian membership function having a fixed standard deviation, σ, and an uncertain mean that takes on values in [m_1, m_2].

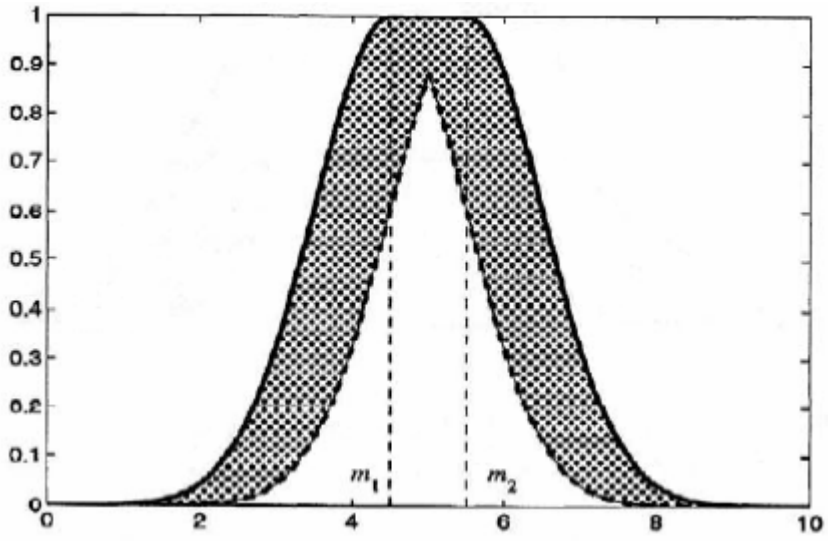

Figure 2-7: FOU for Gaussian primary membership function with uncertain mean

Similarly, let's consider a Gaussian membership function having a fixed mean, m, and uncertain standard deviation that takes on values in [σ_1, σ_2], see Figure 2-8.

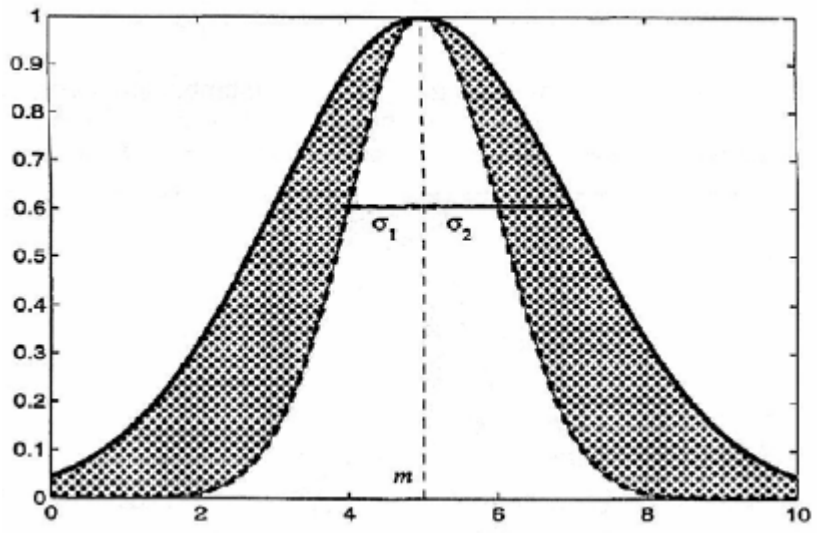

Figure 2-8: FOU for Gaussian primary membership function with uncertain standard deviation

It is to be noted here that both the Gaussian membership functions are interval type-2 because the shading is uniform. Mendel developed the mathematics necessary for building fuzzy logic systems using these two types of Gaussian membership function.

In the following discussion, we will describe the main components of a type-2 fuzzy logic system and see how the uncertainty issues are considered.

2.3.3. Fuzzification in Type-2 Fuzzy Logic System

A fuzzy logic system is considered to be type-2 as long as any one of its antecedent or consequent sets is type-2. A detailed description of all the components of Figure 2-9 is provided by Mendel [49]. In the following discussion, we will discuss fuzzifier for being the most important component from the uncertainty aspect.

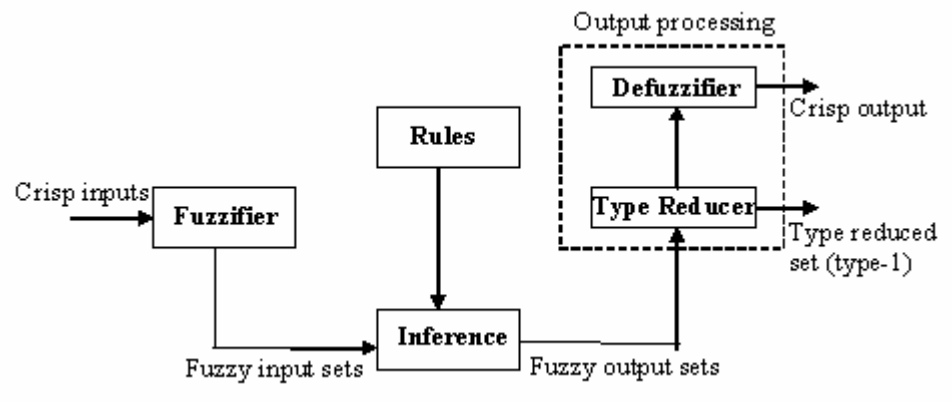

Figure 2-9: Type-2 FLS

Fuzzification is done in two ways- singleton and non-singleton. Singleton fuzzification considers the measurement that activates the FLS to be precise (crisp). Non-singleton considers such input measurements that are imprecise. For a singleton input, the result of fuzzification is a fuzzy singleton i.e. the membership function has a value of 1 only at the input value. On the other hand, conceptually, a non-singleton fuzzifier implies that given input represents a confidence interval with each value in that interval having different (may be same) degree. This non-singleton fuzzification can also be done in two ways: type-1 and type-2 based on the type of fuzzy sets used for fuzzification. Based on different types of fuzzification and different types of antecedent fuzzy sets, Mendel has developed 5 different fuzzy logic systems. Those five different FLS s are:

i. Singleton type-1
ii. Non-singleton type-1
iii. Singleton type-2
iv. Non-singleton type-2 with type-1 inputs
v. Non-singleton type-2 with type-2 inputs

Figure 2-10 shows a pictorial description of these 5 different fuzzy logic systems [49].

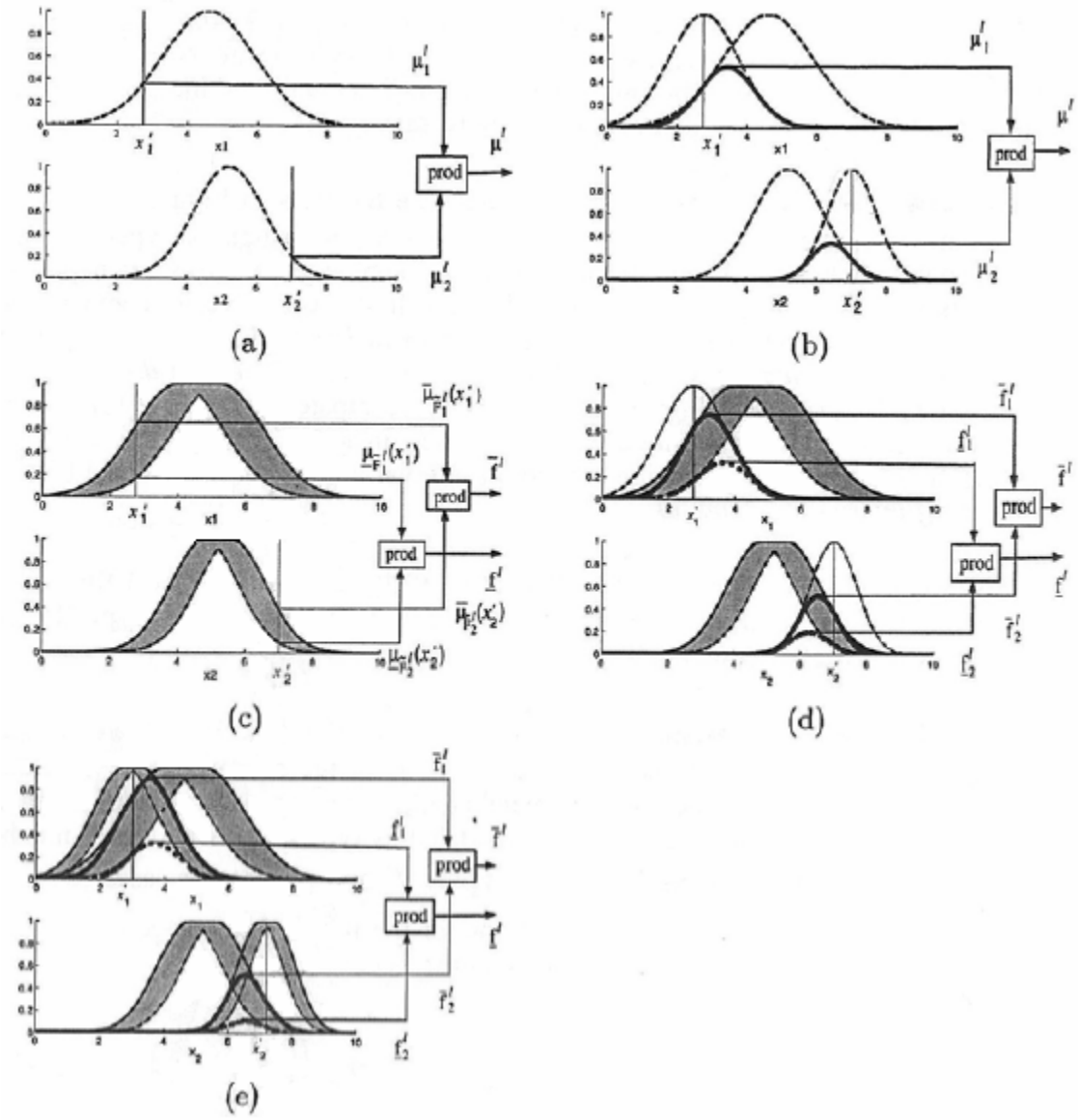

Figure 2-10: Different types of FLS – (a) singleton type-1, (b) non-singleton type-1, (c) singleton type-2, (d) non-singleton type-2 with type-1 inputs, (e) non-singleton type-2 with type-2 inputs.

CHAPTER 3

SOFTWARE SIZE METRICS: A SURVEY

3.1. Introduction

Accurate and reliable software size estimation has a great impact on the software development effort prediction because most methods of predicting effort require an estimate of the size of the intended software system [43][5]. Once the estimate for the software size is available, suitable frameworks/models can be used to relate size to effort [19]. However, producing a reliable size estimate is a difficult task in the sense that estimates obtained at the early stages of development life cycle are inaccurate because not much information of the future system is available at that time [19]. Moreover there are various factors that contribute to the size of software as discussed below. Considering all the factors and incorporating their contribution into the computation process is a challenging task, because no historical data, standard or benchmark is available to validate models and computation processes against.

Fenton and Pfleeger [20] have defined software size as a function of length, functionality and complexity, such that:

$$Size = f(l, f, c)$$

Length is the physical size of the software. It is a common practice in the software engineering community to use the number of lines of code (LOC) as the length of the program source code. But interestingly there is no common practice to calculate this because of the inclusion of blank lines, comments, lines with two or more statements etc. in a program source code. So different people have defined different criteria and the result is that software engineering community has failed to come up with a unanimous standard. Moreover, as mentioned earlier, software size estimates are required early in the life cycle but counting LOC is dependent upon information which is not available until later in the development life cycle. Additionally, the software engineering community has witnessed only a few methods for estimating size in terms of lines of code, by analyzing requirement and design specifications, in early stages of software development e.g., the techniques proposed in [13][65]. Unfortunately, the methods that have been proposed could not get wide acceptance in the software engineering community [15].

Functionality defines usefulness of software or it measures the number of operations and capabilities provided by the software to the user. Researchers believe that the functionality seen by the user provides an indication of the size of the software[3][72].

Complexity, on the other hand, can be construed in following different ways depending on ones perspective [20]:

i. Problem complexity
ii. Algorithmic complexity
iii. Structural complexity
iv. Cognitive complexity

The existing software size metrics in the literature or those in practice reveal that researchers have put efforts to capture structural complexity, where applicable, in formulating size metrics. Since, as the name suggest, structural complexity measures the structure of the software used to implement the algorithm [20], therefore it enables researchers to look at various requirement and design specifications to extract information about the control flow structure, hierarchical structure and modular structure and then formulate the extracted information in some way to produce a software size measure.

Besides length, functionality and complexity; the reuse through inheritance can also be regarded as an aspect of software size [72]. The examples of size metrics that made use of functionality, complexity or reuse through inheritance are Function Point (FP) [3][5], Use Case Point (UCP) [39], Class Point (CP) [19], Predictive Object Point (POP) [72], Feature Point [32], Object Point [36], Fast&&Serious [13] and Vector Size Measure (VSM) [24] etc.

It is desirable that a software size metric should consider all the aspects (dimensions) of size in order to produce a reliable estimate; however, this seems not to be fulfilled by all the existing metrics proposed in the literature. For example, function point and use case point extract information regarding software functionality and can be applied early in the software life cycle but one of the major criticisms on them and their descendants is their inability to address complexity adequately[25][24][25][30][32][70][71]. The consequence of this is disproportional measurement of software size [24]. Only a few metrics have incorporated information regarding functionality, complexity and reuse through inheritance such as POP [72]. Moreover, there are some other attributes, besides software aspects, that must be considered before declaring the usefulness of a size metric. These attributes are discussed in the following section.

3.2. Attributes for Evaluating Software Size Metrics

To the best of our knowledge, there is no standard available to define the satisfactory or acceptance criteria for a software size metric, therefore we have come up with some attributes, based on our literature survey, which can be used to assess the efficacy of the existing and future size metrics. In addition, these attributes can be used to compare and classify different size metrics in a more objective manner. The attributes evaluate metrics from several aspects: aspects of software size are addressed, paradigms targeted are discussed, the views of the software covered by the metric, the mechanism for formulating the extracted information, the input artifacts, weighing scheme adopted, the unit of measure proposed, the mode of operation, the development phase in which metric can be applied, the granularity and whether a metric is reproducible. Therefore these

attributes give a good indication of how much appealing a size metric is, in terms of accuracy, reliability, ease of use, etc. The following subsections discuss these attributes.

3.2.1. Paradigm Coverage

This attribute refers to the paradigm: procedural or object oriented; where the metric can be used. Today's software development relies on object oriented paradigm as the de-facto standard. Therefore size metrics designed to work with the object oriented paradigm would seem more desirable.

3.2.2. Development Phase

This attribute determines at which phase e.g., requirement engineering or design, in the software development life cycle can the metric be applied. The earlier a size metric can be applied the more desirable it may be (if it reasonably considers other attributes) because early estimates play a vital role in making decisions for contract bidding and determining the feasibility of project on cost-benefit analysis [39]. Moreover, effective strategies for planning and tracking the software projects are highly based on the careful and accurate cost and schedule estimation, which in turn are based on size estimate of the future system. This helps in effectively controlling the expensive software development investment, which is of preponderant importance [46][68].

3.2.3. Aspects Coverage

It refers to the aspects considered by the metric in order to produce a size estimate. Actually, the size metric reliability not only depends upon the aspects it covers but also upon the way they are covered. Three aspects that software metric should cover are [72]:

i. Functionality
ii. Complexity and
iii. Reuse through inheritance

The important thing here is to note that metrics do consider length of software while developing a size metric. Actually, some metrics provide a mechanism to predict effort directly by using their outcome e.g., FP [3], UCP [39], whereas some metrics use their outcome to predict length of the intended software e.g., VSM [24]. This length can then be used to produce effort estimates by using some well renowned effort prediction systems, e.g., COCOMO.

The functionality (behavior of object, methods or modules, which determines what a user actually gets) provides a significant piece of information that must be captured in order to provide a reasonable size estimates. However, considering functionality only may lead to misleading estimates, particularly in a well-defined procedural or object oriented software systems [72]. The reason for this is that in addition to functionality, there is level of complexity that results from, for example, heavy communications among the objects or the methods in the system e.g., Coupling, Cohesion, Algorithmic difficulty [72]. This complexity contributes significantly to the size of the software project [72].

The reuse through inheritance is only applicable to the object oriented based software systems. A group of objects that share many similar attributes and behaviors is designed as a base class. Several other classes can be derived from the base class by augmenting some specific attributes and behaviors to the base class while possessing the characteristics of the base class as well. This way inheritance allows a significant reduction of size and hence effort in certain software projects.

3.2.4. Views Coverage

We know that with the help of various diagrams one can capture multiple views of the software characteristics e.g., functional, structural, deployment, and complexity, where each view contributes to size of the software. It is essential to see how many views and how effectively they are considered by a size metric to extract the information related to these views.

3.2.5. Input Artifacts

This attribute reflects the types of artifacts that are used in order to compute a particular metric. There are various types of artifacts available for object oriented software modeling e.g. the Unified Modeling Language (UML) diagrams [77] and the Object Oriented Modeling Technique (OMT) [58]. The procedural based software projects are usually modeled by data flow diagrams [20]. This attribute is mainly used for deciding on the suitability of a metric for a specific project (model). In addition, it can also identify those areas (models) for which there are no metrics available for estimating size. Thus it requires from researchers to put efforts in designing metrics that can be used with those

models. This way software engineers will enjoy more freedom to employ models of their interests.

3.2.6. Granularity

Granularity determines which level, class/object or method is used to gather the information in order to provide a size estimate. In the object oriented environment there are two prominent schools of thought in the determination of object-oriented metrics suitable for size estimation [72]. One employs classes/objects to get useful information e.g., the works of Laranjeira [43] and Jenson and Bartley [31]; whereas the other employs inter and/or intra method interactions to produce size estimates e.g., the works of Chidamber and Kemerer [16], Pittman [54], Lockheed Martin [45] and Minkiewicz [50]. Methods are at deeper level than objects, which means more detailed information about the structure and complexity of a software system can be captured if a metric digs down to the method level. This leads to more reliable and accurate size estimates.

3.2.7. Unit of Measure

Unit of Measure concerns with the unit of measure for software size metric. The effectiveness and acceptability of a software size metric increases if it comes up with some measure that can reflect with high confidence the amount of effort required to develop a unit of that measure e.g., POP [72] and Class Points [19]. This is necessary rather than just desirable because the size estimates form the bases for predicting effort [72], cost and other project related attributes e.g., cost-benefit analysis [39], contract bidding decisions[39], planning and tracking[46][68]. The metrics that produce size

measures without fulfilling this requirement may not be helpful. Moreover, besides inherent factors of the underlying problem that the software is solving, there are various external factors that contribute to the software cost, e.g., the working environment, platform and personnel skills [10]. The examples of size metrics that consider such factors are function point [3], use case point [34]. Therefore it would be appropriate for a size metric to suggest a way as how to combine theses factors with its outcome to produce reliable cost estimates.

3.2.8. Weighing Scheme

As the name suggests this refers to how the various attributes such as method type [72], complexity level of a class [19] etc., of intended software are assigned weights in order to produce some quantitative value that reflects the estimated size of that software. In this context there could be two possibilities [20]:

i. Subjective weighing and
ii. Objective weighing

In the subjective assessment an expert is asked to analyze the requirement/design documents and assign the weights to various software attributes as proposed by the underlying metric. The subjective assessment, as we know, may differ from expert to expert thus contributing to uncertainty in the estimation process. Moreover, in most of the cases such as FP [3], POP [72], Fast&&Serious [13] etc., the weights assigned to the enumerative rating scheme within a particular attribute show abrupt (unsmooth) change that may lead to inaccurate estimates. Consider, for example, the weight assignment to class functionality levels of some arbitrary size measure in Table 3-1. It is crucial to

assign a proper level (Low, Medium etc.) of functionality to a class because there is a significant change in the weight assigned to various levels while there is not much difference in the starting and the ending, or vice versa, limits of the adjacent levels. Furthermore, one must be very careful if such type of information is gathered from some experts because different experts may provide contradicting assessments.

Table 3-1: Weights assignment to different levels of class functionality

Number of Services	Class Functionality	Weight
0-2	Very Low	1
3-5	Low	3
6-8	Medium	4
8-11	High	5
12 and more	Very High	7

In the objective assessment no explicit weights to the software attributes, as considered by a particular metric, are assigned rather the attributes of the intended software are formulated and manipulated in such a way that it results in some quantitative value e.g., the scheme adopted by VSM [24].

3.2.9. Soundness

This attribute determines whether the proper theoretical validation is carried out by satisfying well-known properties necessary for the size metric, such as, desirable size properties specified by Briand *et al.* [11] or formal validation framework defined by Kitchenham *et al.* [40]. Moreover, an empirical validation is also necessary to investigate the usefulness of the proposed metric as a means of estimating the effort of the software

projects. In fact, the software engineering community positively acknowledges a metric only if its usefulness has been proven by means of a validation process [19].

3.2.10. Formalization

This attribute describes how different pieces of extracted information (i.e., constituents of size metric) are combined in the metric to produce a size estimate. Actually, formulating a size metric is not an easy task because different metrics have different constituents. There is no definite way to determine the contribution of each constituent of the metric in estimating the software size. The only possibilities to carry out this task is to perform rigorous statistical analysis of sufficient historical data, e.g., as performed in POP [72], or to consult experts who can provide useful insight on the contribution of each constituent based on their past experience, e.g., as performed in FP [3].

3.2.11. Mode of Operation (MOP)

This attribute determines whether a size metric is applied automatically using some tools or it is manual that is tedious and needs some expert judgment. If it is manual then the attribute further discusses the possibility of whether the computation process can be automated. Actually, the acceptance of a size metric increases if it can be applied automatically because, in many cases, manual evaluation depends on some expert judgment that may lead to uncertainty due to different experts opinions.

3.2.12. Reproducibility

This attribute reflects the ability of the metric to re-produce the same estimate, within an acceptable tolerance limit, when the same input dataset is applied. Assuming the metric is deterministic, it is highly likely that if the process of applying a size metric is automatic then it will reproduce the same value, otherwise if this process is manual then it may lead to some different values. Actually, estimating the size of a software is usually an iterative process which goes along the software life cycle phases. At each phase new characteristics are added to the software, which carry useful information that contributes to the software size. Therefore, it is desirable that the metric should consider this added information and produce the size estimate that reflects this information. If the metric is not reproducible, probably due to some manual evaluation, then it may not be able to give a clear indication of the contribution of the added information on the software size.

3.3. Evaluation of Popular Size Metrics in Practice and Literature

In this section we will evaluate Function Point approaches, Use Case Point approaches, Predictive Object Point, Vector Size Measure, Class Point and Fast&&Serious on the basis of our attributes set.

3.3.1. Function Point Approaches

The Function Point metric was first proposed in 1977 by A. J. Albrecht [3] as a method of measuring software size and productivity. It employs functional and logical entities

such as inputs, outputs, files and inquiries that are believed to relate more closely to the functions performed by the software as compared to other measures, such as lines of code. The advent of Function Point results from the perception that length is misleading, and the amount of functionality inherent in software portrays a better image of software size. The advantages of Function Point are that it can be applied early in software lifecycle and does not expect the specification to conform to the prescripts of particular technique or standard [20]. However, one of the major criticisms on the Function Point analysis is its inability to incorporate complexity adequately in the computation process. This can result in disproportionate measure of software size [24][78].

Several extensions of Function Point were also proposed to incorporate complexity and overcome other issues e.g. it can not be applied to embedded and real time applications [25][24] [25][32][71][78]. MKII Function Points [71] and Feature Point [32] are the most tested and accepted among these alternatives. The problem with MKII approach is that it requires calibration, which may be difficult for such application types for which there is no or little history available, whereas Feature Point became less popular in the Function Point community. Full Function Point [21] and 3D Function Point [78] have shown potential but have not been thoroughly tested in all the environments [24]. In addition, although 3D Function Point captures all three aspects of software size but it only achieves this at a class level, which makes it a good metric for productivity analysis of completed software but less valuable for effort prediction [72]. Finally, several object oriented versions of Function Point have been proposed [5][74]. We evaluate Function Point and its object oriented descendants [29] on the basis of defined attributes in Table 3-2. The

other extensions of Function Point e.g., MKII approach, 3D Function Point etc, can be evaluated in the similar fashion.

Table 3-2: Evaluation of Function Point approaches to software size estimation

	Attributes	Approach
1.	Aspects Coverage	Functionality is captured by counting the items like external inputs, external outputs, external inquiries, external files and internal files.
2.	Input Artifacts	It accepts any software specification that contains the desired information. Some adaptation of function point techniques to object oriented based approaches make use of class diagrams and sequence diagrams. Moreover, some subjective involvement in assessing the technical complexity factors is required.
3.	Paradigm Coverage	The approach can be applied to both procedural and object oriented paradigm.
4.	Granularity	The adapted FP techniques for object oriented base software capture the class level details.
5.	Unit of Measure	The unit of measurement is Function Point (FP), which can form the basis for an effort estimate.
6.	Weighing Scheme	It is subjective but there are some tools that can be used to partially assess various attributes during the computation process, objectively. But the big issue is the subjective choice of weights for calculating unadjusted function points which was derived from IBM experience. These values may not be generalized.
7.	Soundness	It was empirically evaluated.
8.	Formalization	The formulas are derived using regression analysis on actual data, whereas weights to different factors are assigned based on debate and trial.
9.	Mode of Operation	It is manual and can be semi automatic.
10.	Views Coverage	If used for object oriented based software (i.e., adapted FP techniques) then class and sequence diagrams can be used. Although class and sequence diagrams mainly render structural and behavioral information, respectively, but these approaches, somehow, only capture information related to functional view.
11.	Development Phase	The approach can be used at software requirements engineering and design stages.
12.	Reproducibility	The estimates can not be reproduced due to high subjective involvement in assessing the technical complexity factors (TCF).

3.3.2. Use Case Point Approaches

Although object oriented versions of Function Point were proposed but since Function Point itself had its roots in the procedural paradigm, which made it difficult for the software engineers to properly map the concepts to object oriented based software. The result is that Function Point has lost favor in the object oriented based software engineering community and the researchers have started ascertaining for the methods that purely rely on object oriented concepts. As a result the Use Case Point method for sizing and estimating object oriented software projects was developed by Gustav Karner of Objectory (now Rational Software) in 1993 [34]. The method is a prolongation of Function Point Analysis and MK II Function Point Analysis and is based on the same philosophy as these methods i.e., the functionality visualized and experienced by the user is the basis for estimating the size of the software. This allows an early estimate of the software size and consequently effort to be made based on use cases when there is some understanding of the problem domain and system functionality is developed [39]. We discuss Use Case Point Approaches based on our proposed attributes in Table 3-3.

Table 3-3: Evaluation of Use Case Point approaches to software size estimation

	Attributes	Approach
1.	Aspects Coverage	Functionality is addressed by analyzing the complexity of actors and the number of transaction contained in a use case.
2.	Input Artifacts	As the name suggest it utilizes use case diagrams and description. Moreover, some subjective involvement in assessing the technical complexity and environmental factors is required.
3.	Paradigm Coverage	It mainly focuses on object oriented based software development but can also be applied to procedural based software development if the functionality of the software is described through use case diagrams.
4.	Granularity	It captures functionality at the system level which is determined by the interaction of various actors with the system.
5.	Unit of Measure	The unit of measurement is Use Case Point (UCP), which can then be multiplied with suitable number of man-hours to produce the development effort estimate.

6.	Weighing Scheme	Basically it is subjective, but some tools have been proposed to objectively assess various entities in the computation process. Moreover, weights assigned have abrupt changes.
7.	Soundness	Empirical validation on three industrial projects is carried out. Therefore the results obtained can not give sufficient indication of the usefulness of the metric in various application domains.
8.	Formalization	It borrows function point formulas and modifies some factors to produce its own formula. The weights of different factors are also borrowed from FP and experts are asked to provide weights of some additional factors and constants.
9.	Mode of Operation	It is manual and can be semi-automatic. But the semi-automatic estimation requires extra effort in describing use cases as required by a specific tool.
10.	Views Coverage	It utilizes use case models and thus captures functional view.
11.	Development Phase	The approach can be used at requirements engineering and design stages provided that the details regarding technical complexity (expert's opinion) and environmental factors are available.
12.	Reproducibility	In general, the estimates may not be reproduced due to subjective involvement in determining technical complexity and environmental factors. Moreover, if the manual process is used then weighing scheme will be a big hindrance in reproducing the estimates.

3.3.3. Predictive Object Point Metric

The rationale behind devising Predictive Object Point (POP) is the unavailability of such size metric that incorporates all the three dimensions i.e., functionality, complexity and reuse through inheritance, within the computation process. The integration of all the three dimensions based on the analysis of objects and their characteristics distinguishes it from traditional approaches, which are based on the data and procedure model of structured analysis [72]. The evaluation of Predictive Object Point based on our proposed approach is presented in Table 3-4.

Table 3-4: Evaluation of Predictive Object Point metric to software size estimation

	Attributes	Approach
1.	Aspects Coverage	The metric is claimed to cover all the three dimensions but the effectiveness of the way they are captured raises a concern. The functionality is captured using some count of the number of services provided to the system by a particular method and the properties affected rather than taking into consideration the nature of the services into account. This is crucial because the provided service may be a simple value returned to a calling function, a simple output on the console or it may require some database modification or modification of some file etc. The same is true with the complexity which highly depends upon the nature of functionality (different functionalities require different level of complexities) and one simply can not measure the complexity of methods by their types only.
2.	Input Artifacts	To compute WMC, the metric highly relies on source code to gather the required information. But one can also use activity diagrams or some specialized ways to gather this, which is only possible when the method/attribute relationship is documented informally in the activity diagrams. In addition, the information regarding inheritance can be extracted using class diagrams.
3.	Paradigm Coverage	It covers object oriented paradigm.
4.	Granularity	It mainly addresses method level details by employing Weighted Methods Per Class (WMC) measure whereas the class level details related to inheritance are also considered. One may have reservations regarding the way WMC is computed and it needs further improvement in the computation process.
5.	Unit of Measure	The unit of measure is Predictive Object Point (POP). This technique does not discuss how POP can be related to the development effort.
6.	Weighing Scheme	It is objective but assigns weights that have abrupt changes within complexity levels.
7.	Soundness	There is no clue regarding theoretical or empirical validation. Although the proposal discusses some empirical verification but results are not published in the literature.
8.	Formalization	The regression analysis on some historical data is performed to determine various coefficients used in the metric. There is still a need to perform rigorous analysis on sufficient historical datasets that cover different application domains.
9.	Mode of Operation	It requires manual assessment. Moreover, there is room to seek the possibility of automating the computation process.
10.	Views Coverage	The metric highly relies on information that may be extracted from activity diagrams, only if they contain method/attribute relationship. Thus it captures information related to methods' behavior.
11.	Development Phase	The metric can be employed at the detailed design level when the algorithms are designed. It is also possible to apply this at the high level design phase if the method/attribute relationship can be captured informally using activity diagrams or some specialized ways.

12.	Reproducibility	The estimates can be reproduced because subjective opinion is not required in the computation process.

3.3.4. Vector Size Measure: A Vector-Based Approach to Software Size Measurement and Effort Estimation

The uniqueness of Vector Size Measure (VSM) arises from the fact that it adopts rather vector based approach than traditional analytical or regression based approaches in formulating functionality and complexity to produce size measure and to classify software systems. It claims to incorporate both functionality and problem complexity[1] in a balanced and orthogonal manner; whereas, length can be derived from functionality and problem complexity. In order to estimate effort, VSM is used as the input to a Vector Prediction Model (VPM), which is proposed by Hastings et al [24]. The purpose of VPM is to estimate development effort early in software lifecycle. Evaluation of VSM on the basis of proposed attributes is provided in Table 3-5.

Table 3-5: Evaluation of Vector Size Measure to software size estimation

	Attributes	Approach
1.	Aspects Coverage	Functionality and complexity are captured by considering methods syntactic and semantics, respectively, after converting the desired information into algebraic specification language. It has adapted Halstead's concept of operators and operands, called as OPs, to define its atomic unit. The functionality is captured through methods signatures which mean the nature of functionality is not considered. In addition, the way the complexity is incorporated into computation may not reflect the effort needed in developing the software because the nature of

[1] The other size metrics presented in the literature usually capture structural complexity as the complexity of the intended software system. It is the uniqueness of this metric that it captures problem complexity, where problem complexity represents the underlying semantics of the software system.

		complexity highly depends upon inter-object communication, methods internal details and the way various resources are utilized to produce the desired functionality.
2.	Input Artifacts	It requires the software be represented in Algebraic Specification Language (ASL), which is based on Abstract Data Types (ADT). But one may ascertain the possibility of gathering the required information from UML Object Constraint Language (OCL).
3.	Paradigm Coverage	The metric is initially intended for object oriented based software but one may also look at the possibility of applying the approach to procedural paradigms.
4.	Granularity	The methods signatures and semantics are considered without digging down to methods internals details. The class level information is not addressed.
5.	Unit of Measure	The unit of measure is OP. The approach has presented the way the OPs can be related to development effort.
6.	Weighing Scheme	There is no weighing scheme.
7.	Soundness	It is theoretical validated against Kitchenham *et al*. formal validation framework [40]. Moreover, the empirical validation reveals that it can be used to predict development effort within ±20 percent across a range of applications.
8.	Formalization	It uses a vector based approach to determine size as a function of functionality and problem complexity. Size is measured in terms of magnitude and gradient. The magnitude considers functionality and problem complexity in a balanced and orthogonal manner. The gradient is a ratio of problem complexity and functionality that measures the relative dimensions of systems.
9.	Mode of Operation	One may look at the possibility of automating the process of counting the OPs from software specifications.
10.	Views Coverage	In its present form the metric does not consider UML models but if desired then use case and state transition diagrams can be used.
11.	Development Phase	The metric can be used at requirements engineering and design level.
12.	Reproducibility	The estimates can easily be reproduced because this technique is independent of weighing scheme.

3.3.5. Class Point Metric

Class Point method basically measures functionality by quantifying classes analogous to the function counting performed by the FP measure. As we know that the basic building blocks in the procedural paradigm are functions and procedures; whereas, in object

oriented paradigm, classes are the logical building blocks, which correspond to real-world objects that interact with each others [19]. Therefore rather than analyzing the functionality at the system level, this metric considers the services required to and from each class to produce a size measure. We discuss Class Point metric on the basis of our proposed attributes in Table 3-6.

Table 3-6: Evaluation of Class Point metric to software size estimation

	Attributes	Approach
1.	Aspects Coverage	Based on the services required to and from each class, the complexity is computed. Moreover, the complexity of each class on the basis of number of attributes is also taken into consideration.
2.	Input Artifacts	The metric relies on software source code to extract required information. Moreover, one can look at the prospects of gathering the information from design models (e.g., sequence diagrams) because the information required to measure class points can be easily extracted from these models.
3.	Paradigm Coverage	It covers object oriented paradigm.
4.	Granularity	It covers class level details only. The Number of External Methods (NEM) and the Number of Services Requested (NSR) are used to capture the complexity of each class for CP1, whereas the Number of Attributes (NOA) measure is taken into account to capture the complexity of each class for CP2.
5.	Unit of Measure	The unit of measure is Class Point. It reflects size by considering both software internal details and external factors. This makes it a prospective candidate for producing good estimates of the required development effort.
6.	Weighing Scheme	It is highly subjective and assigns weights that have abrupt changes within complexity levels. This can be noticed in the computation of Total Unadjusted Class Points and Degree of Influence.
7.	Soundness	It is theoretically validated against desirable size properties defined by Briand *et al.* [11]. The initial empirical validation is performed on 40 students' projects. The empirical validation reveals that CP2 predicts effort better than CP1.
8.	Formalization	The experts' knowledge is incorporated in determining the contribution of various factors.
9.	Mode of Operation	It is manual. Moreover, it is not straight forward to seek the possibility of automating the computation process due to high subjective involvement in assigning the weights to various class complexity levels.
10.	Views Coverage	The metric is intended to estimate size from design documents. In this context one can employ class diagrams and sequence diagrams to gather desired structural and behavioral information, respectively.

11.	Development Phase	It works at the software design level.
12.	Reproducibility	The estimates can not be reproduced due to greater subjective involvement in weighting.

3.3.6. Fast&&Serious: A UML Based Metric for Size Estimation

The UML diagrams have become a de-facto standard for modeling object oriented based software. The lack of fully automated methods to extract such information from UML diagrams that paints a better picture of the size of the object oriented based software encouraged researchers to invest effort. Some methods were proposed but they usually considered only a few UML diagrams to extract information regarding various aspects of size [72][74]. They usually adapted the function point approach for object oriented based software e.g., [74], or automated use case point approach e.g., [42]. Carbone and Santucci [13] have come up with Fast&&Serious method that analyzes various UML diagrams automatically and produces size estimate in source lines of code. We evaluate their metric on the basis of our attributes in Table 3-7.

Table 3-7: Evaluation of Fast&&Serious metric to software size estimation

	Attributes	Approach
1.	Aspects Coverage	The metric mainly focuses on analyzing the complexity and inheritance. It has also addressed functionality in a loose sense when assessing the complexity using use case diagrams.
2.	Input Artifacts	The sources of information are various UML diagrams e.g., use case diagrams, class diagrams etc.
3.	Paradigm Coverage	Object oriented paradigm is the target of this approach.
4.	Granularity	The metric addresses class and method level details. The amount of communication involved for a particular class is extracted through sequence diagrams whereas the method level complexity is assessed using their signatures.
5.	Unit of Measure	The basic unit of measure is Class Point (CP), which can be used to produce estimates for the lines of code using a formula. There is no

		justification or rationale for the formula used to convert CPs into lines of code.
6.	Weighing Scheme	It is objective but the weights assigned are abrupt. Moreover the basis for various tables used in different computations is not explained. Moreover the values contained in these tables have abrupt changes.
7.	Soundness	It is not theoretically validated. Moreover, only a single empirical analysis is provided which by no means is sufficient.
8.	Formalization	It does not discuss the rationale behind combining various factors to produce formulas that are used for computation. Moreover, there is no justification or rationale for the formula used to convert CPs into lines of code.
9.	Mode of Operation	It is automatic.
10.	Views Coverage	The metric addresses structural, behavioral and functional views by making use of class, sequence, state transition and use case diagrams.
11.	Development Phase	It can be applied at requirements engineering and design levels.
12.	Reproducibility	The estimates can be reproduced because of objective weighing.

3.4. Conclusions and Future Directions

The critical survey of the existing software size metrics, using our attributes set, reveals some shortcomings. These shortcomings include, but are not limited to the following

i. The majority of the approaches do not capture all the dimensions that provide different aspects of software size. In addition, different metrics have adopted different ideas of capturing functionality and complexity. In many cases it is found that they lack the capability to capture functionality by considering the nature of functionality e.g., Predictive Object Point [72] assigns weight by considering method type rather than considering what actually the method is providing.

ii. The metrics, in general (excluding FAST&&SERIOUS), have explored the structural aspects of software specifications. They have considered information

like associations between the classes, inheritance hierarchy, methods' signatures, the attributes referenced within the methods, and use of information hiding. The modeling of the dynamic aspects of the software is not addressed satisfactorily. For instance, the information that is presented in sequence diagrams, activity diagrams and state chart diagrams etc.

iii. In general, the metrics are partially formalized and in many cases the rationale behind combining extracted information to produce a formula is improperly defined. It seems that they are formalized using intuition rather than investigating how different pieces of information contribute to software size.

iv. Validating the proposed metric is not addressed properly. In majority of the cases there is no theoretical validation. They rely mostly on simple empirical validation lacking significant data and statistical analysis.

v. Different metrics have different units of measure and there are few techniques that correlate size measure to software effort. Some techniques provide a correlation based on past experience which may not be applicable at all times and in all domains.

vi. The subjectivity involved in various metrics leads to uncertainty. This uncertainty leads to uncertainty in the effort estimation. Moreover, different ways of calculating the same measure, e.g., Function Point, also contributes to uncertainty.

vii. Some techniques can not be automated which may lead to their unacceptability in the software engineering community.

viii. There are only a few techniques that can be applied at requirements engineering stage of the software development life cycle. The researchers must ascertain the

prospects of enhancing the other techniques so that they can be used early in life cycle, if at all possible.

Future research should try to investigate solutions for these issues. In the next chapters we will provide solutions to incorporate uncertainties, which arise due to the existence of various size measures and deficiencies associated with them, while predicting development effort.

CHAPTER 4

LITERATURE REVIEW

The development of effective software effort prediction methods has been a research target for quite a long time [1]. The methods presented so far can be broadly classified in two main categories, namely algorithmic and non-algorithmic effort estimation [63][22] .

4.1. Algorithmic Effort Estimation Models

An algorithmic effort estimation model involves one or more mathematical formulas which, typically, have been derived through statistical data analysis e.g., regression analysis) and calibration [4][17][75]. Examples of popular effort estimation models include Constructive Cost Model (COCOMO) [10], Constructive Cost Model II (COCOMO II) [9], and Software Life Cycle Management (SLIM) [55]. Detailed critical surveys on algorithmic effort estimation approaches are presented in [22][69][1] . Following are the major drawbacks of the algorithmic approaches, which contributed to

the number of studies exploring non-algorithmic models, e.g., artificial neural networks and fuzzy logic.

i. The inability to handle categorical data (data that are defined by a range of values) [62].

ii. The lack of providing reasoning capabilities, that is the inability to draw conclusions and make judgments based on available data [63][2].

iii. Models (i.e., formulas) may be company specific and may not be applicable for software development in general [22].

iv. The use of historical data has some limitations because attributes and relationships used to predict software development effort can vary over time, and/or differ among software development environments [67].

v. Besides software size, there are many other factors, i.e., cost drivers [10] that can play a vital role in predicting software development effort. It is very difficult (if not impossible) for a model to consider all such factors within a mathematical model [73]. This associates a great deal of uncertainty to the estimated effort.

4.2. Non-Algorithmic Effort Estimation Models

Non-algorithmic techniques include Price-to-Win [10], Parkinson [10], Expert Judgment [2][10] and Machine Learning approaches [64]. Machine Learning approaches include techniques that are based on Fuzzy Logic, Regression Trees, Analogy, Rule Induction, Bayesian Belief Networks and Neural Networks [64]. Among these techniques, the soft computing-based category includes the techniques that are based on artificial neural networks, fuzzy logic models, Bayesian belief networks and genetic algorithms.; refer to

[62][63] for a brief description of these techniques. This thesis work utilizes Fuzzy Logic System (FLS), which is regarded as Neuro-Fuzzy system; to develop a framework therefore we will explain this more in the following discussion.

Neuro-Fuzzy, a union of neural networks and fuzzy logic, was first used for cost estimation by Hodgkinson and Garratt [26]. A neurofuzzy system combines the linguistic attributes of a fuzzy system with the learning and modeling capabilities of a neural network to produce transparent and adaptive system. Since fuzzy logic, with its powerful linguistic representation, can represent imprecision and uncertainty in inputs and outputs [63]: accordingly, some researchers have tried to fuzzify some of the existing algorithmic models to handle imprecision surrounding such models, where as handling uncertainty using fuzzy logic is not addressed yet. Zonglian and Xihui [83] made first attempt to fuzzify various aspects of the most successfully and widely used model, COCOMO [38] by proposing f-COCOMO, where they introduced how to incorporate fuzzy logic techniques in COCOMO. Idri and Abran [27] applied fuzzy logic to the cost drivers of intermediate COCOMO model, and after some time in [51] Musilek et al introduced fuzzy logic to represent *mode* and *size* as inputs to COCOMO model. The reader is advised to consult the work by Saliu [62][63], and Hareton and Zhang [22] for more details on these techniques.

There are also the works presented by Ahmed *et al*. [2] and Liang and Noore [73]. We discuss their works in more details due to their great relevancy to the work of this thesis.

In what follows, we will briefly discuss their works and identify the causes of uncertainty.

Ahmed *et al.* [2] carried out an excellent effort when they presented an adaptive fuzzy logic based framework for intermediate COCOMO81 effort prediction model. The framework includes two stages, one for each component of the COCOMO model as given in equation1, see Figure 4-1. The framework can then integrate both the components into a single framework that allows fuzzy rules generation, rules training and expert knowledge incorporation. Due to the unavailability of the effort multipliers data for cost drivers, they only provided the experimental results for the basic part i.e., the nominal effort.

$$PM_{nominal} = A \times [Size]^{B} \tag{4-1}$$

$$PM_{total} = PM_{no\min al} \times \prod_{i=1}^{15} EM_i \tag{4-2}$$

The framework is effective in the sense that it provides better effort estimates than COCOMO itself but on the other hand it is not that effective in the sense that it is incapable of handling uncertainty. As discussed in Chapter 1, uncertainty comes into play when two or more different metrics (or the same metric with subjectivity involved) compute the same quantity (e.g., size), but due to different underlying understanding of the problem they produce different outcomes e.g., different measuring approaches for LOC and Function Point [1][29]. In the same fashion we can have numerous approaches for other popular size metrics such as Use Case Points [39], Class Points [19], Predictive

Object Points [72], Vector Based approach for size measurement [23] etc. This situation is very interesting and at the same time very challenging for the project manager as to which size measure should he use to come up with effective effort prediction for the project. If the project manager is willing to use the framework proposed by Ahmed *et al.* than he would be in trouble because, the framework is incapable of handling noisy or uncertain size measures. The incapability comes from the fact that the basis for the framework is type-1 fuzzy logic system which itself is designed to tackle the imprecision in the fuzzy sets and not the uncertainty, due to noisy measurement or the different subjective opinions [49].

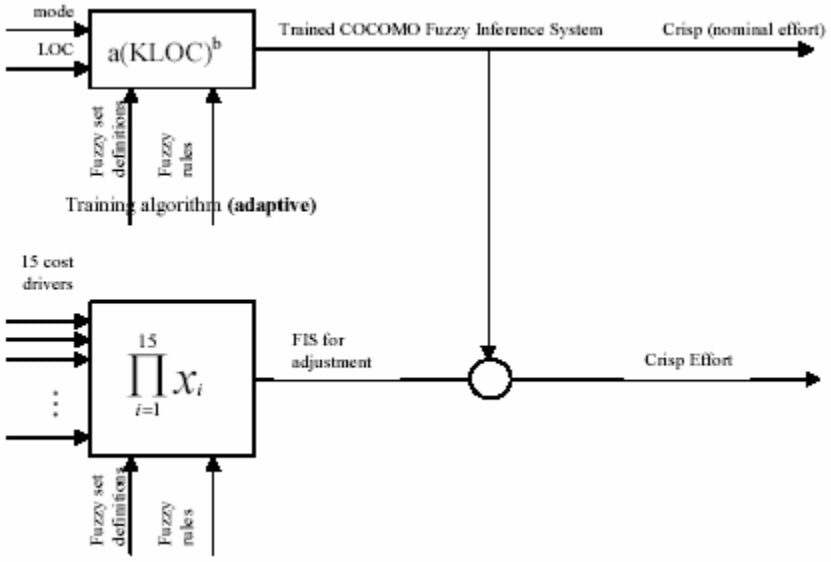

Figure 4-1: Adaptive Fuzzy Logic based framework proposed by Saliu *et. al.*

Besides noisy and uncertain size measures there are other factors like laziness and ignorance that contribute to uncertainty in software development effort prediction. The laziness and ignorance come into play when someone does not consider all the factors that can affect the effort prediction accuracy, e.g., the framework proposed by Liang and

Noore [73]. They suggest that out of 17 only 6 cost drivers: RELY, CPLX, TIME, ACAP, PCAP and PCON are enough to estimate the impact of cost drivers on the software development effort when using COCOMO-II model. They have provided following reasons for adopting this approach.

i. It is not always possible to obtain all the 17 effort multipliers with any degree of confidence.

ii. All cost drivers are not equally important and they may not be totally independent with each other, which lead to difficulty in quantifying their relationship.

They have also proposed a Type-1 fuzzy logic based multistage framework to incorporate these cost drivers in COCOMO-II model for effort prediction, see Figure 4-2.

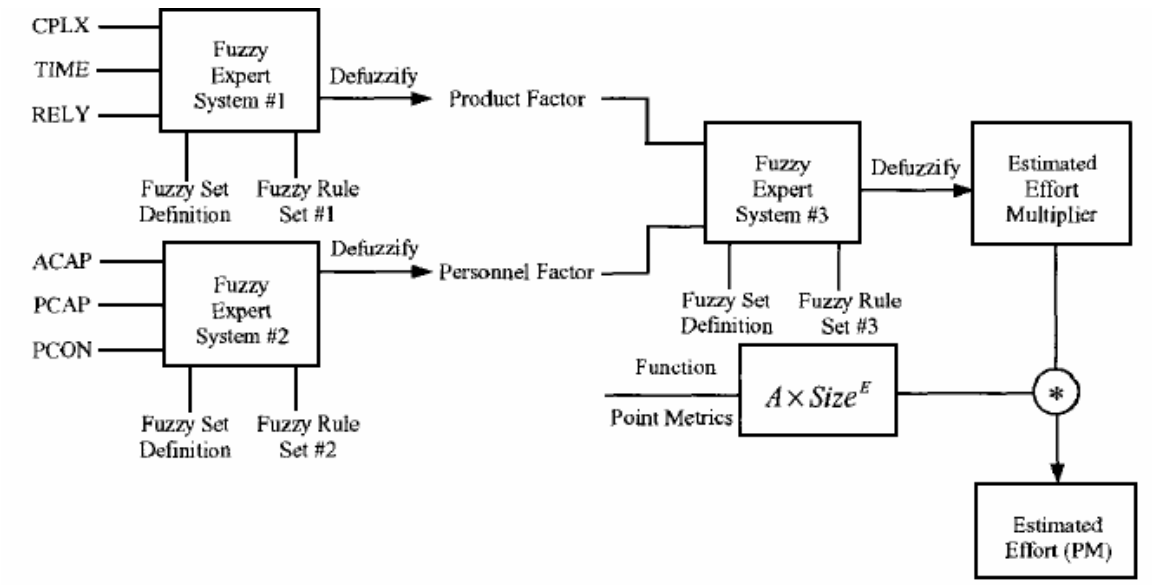

Figure 4-2: Fuzzy Logic based multistage framework proposed by Liang and Noore

Liang and Noore claim that the results obtained by only considering the proposed 6 cost drivers are comparable to the results of COCOMO but the approach is incomplete in the sense that when one gets lazy (although this reduces the burden from the software development team to pay attention to all the cost driving factors and hence reducing significant amount of effort) to incorporate the other 11 cost drivers (see Table 2-3) because of their claim or because of the unavailability of all the cost drivers, it means one is introducing uncertainty by not including these cost drivers. Due to this uncertainty one will get the same effort adjustment factor (EAF) for the two different projects that have the same effort multipliers for the selected 6 cost drivers and different effort multipliers for the rest of the cost drivers. Thus the lack of the approach to take into consideration the impact of introduced uncertainty may not be acceptable for the COCOMO community (the developers and the users of COCOMO model). Our perception is that the prediction accuracy can be improved if a framework using type-2 fuzzy logic system is utilized.

The literature survey reveals that to date no software development effort prediction framework is capable of handling imprecision and different sources of uncertainty, as described in Chapter 1, within a single framework. Most researchers have focused on developing effort prediction systems that can handle the inherent imprecision [61][83][2]. According to a recent and comprehensive survey on soft computing based effort prediction models, carried out by Saliu and Ahmed [62], existing soft computing based effort prediction models lack the ability to handle various sources of uncertainty. Although some approaches presented in the literature try to handle uncertainty using Bayesian Belief Networks (BBNs) [33][6]. The problems with such approaches are that

they are not transparent (i.e., incapable of incorporating expert's knowledge) and incapable of handling uncertainty by modeling the sources of uncertainty, as discussed in Chapter1.

4.3. A New Classification of Effort Prediction Techniques

Besides categorizing the prediction techniques into algorithmic and non-algorithmic, a new classification can also be made if we look at the inputs acquisition and whether the prediction systems can handle uncertainty. In the earlier chapters we have already shed some light on the data uncertainty issue i.e., the sources of uncertainty, now we will discuss inputs acquisition.

Inputs can be acquired in two different ways. One possibility is to use some metric to compute corresponding value for a particular attribute e.g., prediction of size by using different size metrics like Function Point [5], Class Point[19] or Predictive Object Point[72]. This value is a crisp (precise) quantity may be some integer or fractional depends upon the metric that has been used. The other possibility is to ask some expert for his opinion regarding some attribute by providing required specification. It may not be always possible for experts to provide crisp values as inputs for internal attributes by analyzing the requirement/design specifications and using their experience. They like to provide a range of values (imprecise) that depicts their interval of confidence. The range of values may be a simple interval set (i.e., constant values in the y dimension of all the points that lie between two values in the x dimension, see Figure 4-3 that represents an

interval between x_1 and x_2) or it can be in some functional form e.g., triangular or Gaussian. The former is called as Singleton and the latter is knows as Non-singleton input. Therefore we can classify the effort prediction techniques into four different categories which are:

i. Singleton and certain
ii. Singleton and uncertain
iii. Non-singleton and certain
iv. Non-singleton and uncertain

Among the existing prediction techniques regression based models, COCOMO and FLS based techniques such as those presented by Ahmed *et al.* [2] and, Lian and Noore [73] can be classified under singleton and certain. These techniques only accept singleton inputs and can not handle uncertainty. Techniques based on Bayesian Belief Networks e.g., [6] and the framework presented by Rahman [56] can be put under singleton and uncertain because of their ability to handle uncertainty. Unfortunately, there is no single example of the last two categories in the literature.

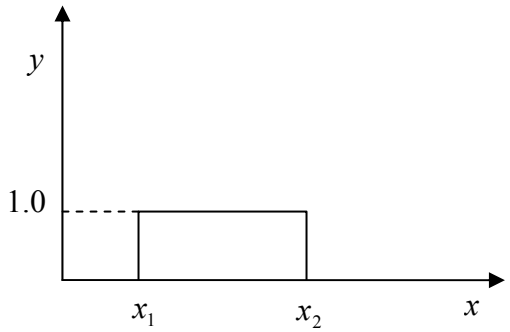

Figure 4-3: An example of interval set between x_1 and x_2

CHAPTER 5

A FLS-BASED FRAMEWORK FOR HANDLING IMPRECISION AND UNCERTAINTY IN EFFORT PREDICTION

As a result of our survey, we can see that there is a need for effort prediction framework that can handle imprecision and uncertainty surrounding the prediction process and deal with the nature[2] of inputs. This chapter provides a solution for handling the nature of inputs, the imprecision and the various sources of uncertainty that are involved when predicting effort. In this quest, we have highlighted sources of uncertainty in the work presented by Ahmed *el al* [2], and Liang and Noore [73]. In order to encounter the impact of uncertainty on the prediction framework proposed by them, we propose to incorporate

[2] It means whether inputs to the prediction system are singleton or non-singleton.

the concept of type-2 fuzzy logic system in their frameworks, which is known to provide solutions for the problems discussed, refer to Section 2.3. The experimental results obtained using type-1 and type-2 systems strengthened our intuition that type-2 system outperforms type-1 system.

5.1. Mapping Uncertainties in Software Quality Model and FLS

We have discussed, in Section 2.3, Mendel's approach to deal with uncertainty in fuzzy logic systems. Rahman [56] has presented a mapping between the sources of uncertainty in generic software quality models and the sources of uncertainty in type-2 fuzzy logic systems proposed by Mendel, see Table 5-1.

Table 5-1: Uncertainties in FLS and software quality models

Uncertainties in Software Quality Model	**Uncertainties in FLS**	**Example**
Linguistic Assessment	Meaning of the word	Expert judgment on the measurement of an internal attribute
Linguistic Relationship	Consequent	How the internal attributes contribute to the external attribute
Numerical Assessment	Measurement to activate FLS	Different metrics to measure a particular attribute e.g., various size metrics
Numerical Relationship	Data to build FLS	Rely only on historical data to build a model

Uncertainties in software quality models have already been discussed in detail in Section 1.1. Here we apply the above mapping to software effort prediction in the context of fuzzy logic systems.

5.2. Solution Approach Using Type-2 Fuzzy Logic System

A general architecture of a type-2 FLS is already discussed in Section 2.3. In the following we will look at the major components of our type-2 FLS based solution for effort prediction. The sequel discusses:

i. Fuzzy sets definition
ii. Rule base formulation
iii. Training
iv. Validation of Prediction system

A flow chart that describes how these components interact with each other is presented in Figure 5-1.

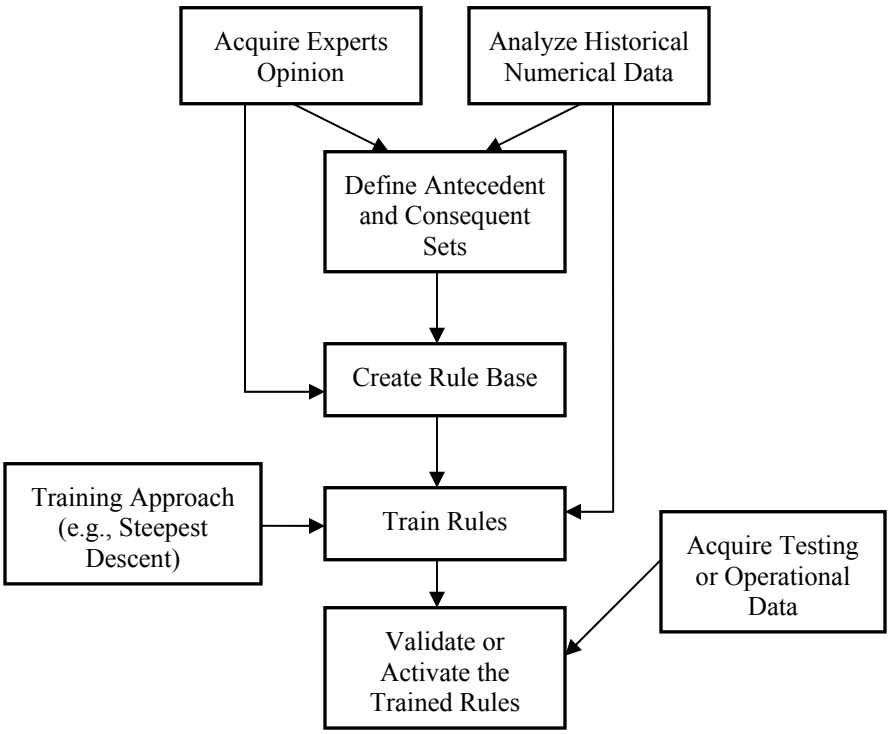

Figure 5-1: Interaction of various components of FLS to produce a prediction system

5.2.1. Antecedent Fuzzy Sets

In building a FLS we divide the whole range of all the internal and external attributes into several fuzzy sets. This fuzzy set classification can be obtained by the experts or the analysis of numerical data sets. As we have already mentioned, different experts may provide different assessments, based on their past experience, regarding a particular fuzzy set (e.g., LOW) range of a specific internal attribute, e.g., size, of intended software by

considering requirement or design documents. Some may say that [0, 25] KDSI[3] is LOW, some may define [0, 30] KDSI as LOW, some may say [0, 28] KDSI as LOW and so on. This causes uncertainty, as to which definition is sounder to consider when one wants to define antecedents while developing FLS. This situation seems problematic but at the same time it is interesting and advantageous as Mendel himself says [49]:

> *"Uncertainty is good in that it lets people make decisions (albeit conservative ones)."*

This statement seems to be supported by Klir and Wierman, when they state [49][41]:

> *"Uncertainty has a pivotal role in any efforts to maximize the usefulness of systems models. ... Uncertainty becomes very valuable when considered in connection to the other characteristics of systems models: a slight increase in uncertainty may often significantly reduce complexity and, at the same time, increase credibility of the model. Uncertainty is thus an important commodity in the modeling business, a commodity which can be traded for gains in the other essential characteristics of models."*

This observation led Mendel to come up with the idea of type-2 fuzzy sets, which enables us to model uncertainty, caused due to different experts' opinion as just discussed, in the FLS by blurring the antecedents' boundaries and defining the footprint of uncertainty (FOU).

---

[3] KDSI as a measure of lines of code (LOC) is only selected as an example. One can use other size metrics as well e.g., class points, function points, predictive object points etc.

Similarly, obtaining fuzzy set classification through analysis of numerical data can cause uncertainty if the data contains more than one metric value for a particular internal attribute. This uncertainty should be reflected in the antecedents using type-2 fuzzy sets so that the impact of uncertainty can be propagated to the outputs through FLS inference engine. An example of type-2 antecedent fuzzy sets is shown in Figure 5-2. In this figure, the range of an antecedent '*x*' is divided into five fuzzy sets out of which two are shouldered.

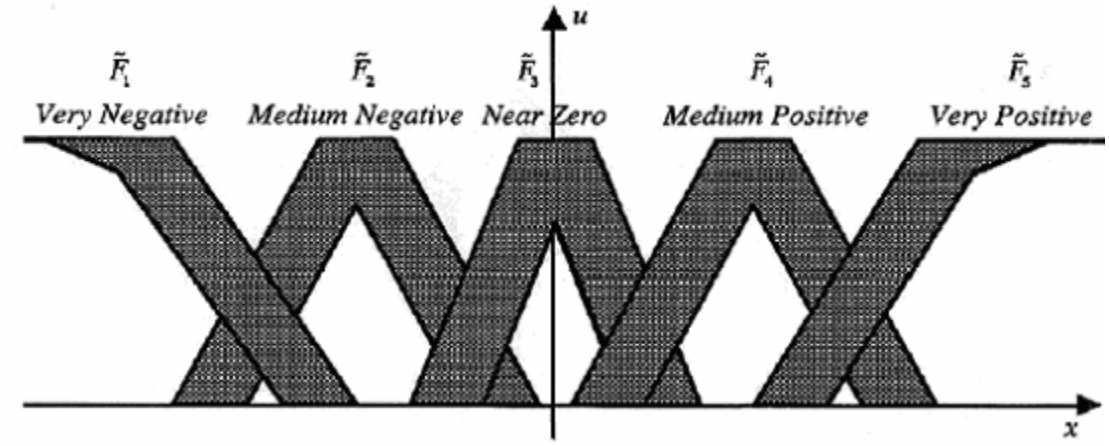

Figure 5-2: An example of antecedent fuzzy sets for some attribute *x*

5.2.2. Consequent Fuzzy Sets

Uncertainty in consequent arises when two or more experts, based on their experience, relate the impact of the same antecedent fuzzy set on more than one consequent fuzzy set using fuzzy rules. In order to handle this situation Mendel has proposed three possibilities [49][56]:

i. Keep the response chosen by the largest number of experts.

ii. Find a weighted average of rule consequents for each rule

iii. Preserve the distributions of the expert responses for each rule

Mendel has opted for the second solution and derived the defuzzification method for handling uncertainty in the consequent. In this thesis, the consequents are defined by all the possibilities of the combinations of fuzzy sets in the antecedents. Moreover, we assume that all the rules are equally probable therefore all rule consequents are equally weighted.

5.2.3. Fuzzy Rule Base

Using type-1 FLS, as proposed by Ahmed *et al.* [2], we generate IF-THEN rules of the following form:

$$IF\ MODE\ is\ M_j\ AND\ SIZE\ is\ S_i\ THEN\ EFFORT\ is\ C_{ji}$$

As mentioned earlier, the distinction between type-1 and type-2 is associated with the nature of the membership functions, which is not essential when forming the rules [49]. The structure of the rules remains exactly the same in the type-2 case, but now the sets involved are type-2:

$$IF\ MODE\ is\ \widetilde{M}_j\ AND\ SIZE\ is\ \widetilde{S}_i\ THEN\ EFFORT\ is\ \widetilde{C}_{ji}$$

These rules are then used to derive the fuzzy inference engine[4].

---

[4] The discussion here is only concerned with handling imprecision and uncertainty in software effort prediction models using type-2 FLS as described by Mendel [49] and employing the framework by Rahman [56]. A detailed derivation of the fuzzy inference engine and defuzzification methods can be found in Mendel's book [49].

5.2.4. Training FLS

After setting up the antecedents and the consequent fuzzy sets by incorporating the uncertainty through type-2 representation and defining the rules using them, the next step is to train the parameters. Training the parameters is desired to refine the rule-based linguistic relationships provided by the experts using the available historical data. That is to say that we refine the linguistic relationships by using the historical numerical data, which have been observed over a period of past experiences. In order to accomplish this we need historical data regarding internal and external attributes from past projects. As we have already discussed in Section 1.1, the problem here is that there may be more than one metrics for a particular internal attribute, e.g., various size metrics. This input noise gives rise to uncertainty and makes it difficult to select a single metric. This situation can be handled if one uses non-singleton fuzzification by properly defining the parameters of the input membership functions [49].

5.2.5. Validating FLS

Finally, we need data to activate FLS. This data may be testing data to validate the performance of FLS or the operational data gathered during developing the software. Since the noise has already been taken care of during training and the parameters are already tuned, therefore one can use type-1 non-singleton fuzzification [56].

5.3. Proposed Framework

The previous discussion has given an insight into the methodology that has been adopted to handle imprecision and uncertainty in the FLS based effort prediction framework. The proposed framework[5] is depicted in Figure 5-3. As can be seen from figure, this framework consists of two stages: nominal effort prediction and effort adjustment factor (EAF) prediction. The outputs of both the stages are multiplied to produce crisp effort. In this section, we will explicitly discuss the specific details as how to use the framework to initialize, formulate, train and validate the nominal effort prediction systems. In the next section, we will discuss how EAF prediction stage can be used to handle uncertainty that arises due to laziness/ignorance.

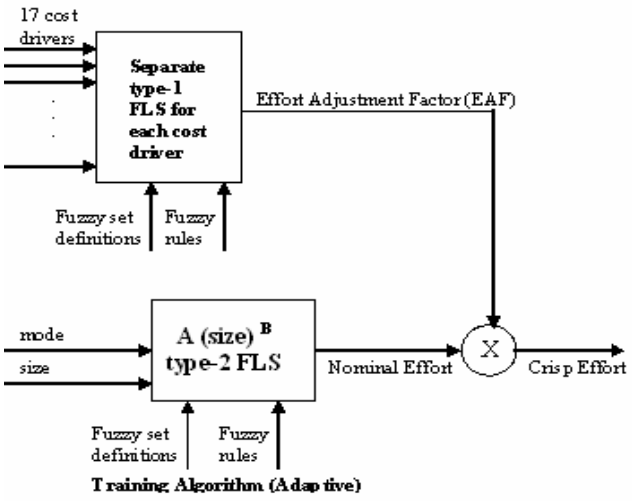

Figure 5-3: Adaptive type-2 FLS based framework for effort prediction.

---

[5] The architecture of this framework is the same as the one proposed by Ahmed *et al [2]*. But this framework contains different FLS, training algorithm and membership functions etc., which are necessary to handle uncertainty besides imprecision.

5.3.1. Initializing the System

Initializing a FLS means initialization of its antecedents, consequents and inputs. In the effort prediction framework proposed by Ahmed *et al.* [2], we have two internal attributes: *mode* and *size*, and an external attribute: *effort*. In this framework internal attributes i.e., mode and size, play the role of antecedents and the external attribute i.e., effort, is termed as consequent. As we have mentioned earlier in Section 5.2.1, there could be two ways to obtain definition of antecedents: through experts' opinion or through analysis of numerical data.

In the case of numerical data, we need to define and initialize various components of the type-2 FLS [49]:

i. Antecedents
ii. Consequent
iii. Inputs

In this context we consider our antecedents and consequent to be type-2 Gaussian with uncertain mean. The input membership functions will be type-1 Gaussian for singleton and type-2 Gaussian with uncertain standard deviation for non-singleton inputs.

After defining the types of the membership functions, we divide the antecedents' intervals into suitable number of fuzzy sets. In our framework, we assume three fuzzy

sets for *mode* as depicted in Figure 5-4. The number of fuzzy sets for size is variable[6], see Figure 5-5. Moreover, the tails of each fuzzy set lie at the mean (M) of adjacent fuzzy sets. This is just to make sure that adjacent fuzzy sets overlap initially. This overlap will exploit the power of fuzzy logic to help us handle software projects that fall between two fuzzy sets intervals [63]. In addition, the two fuzzy sets that lie at the interval boundaries are shouldered.

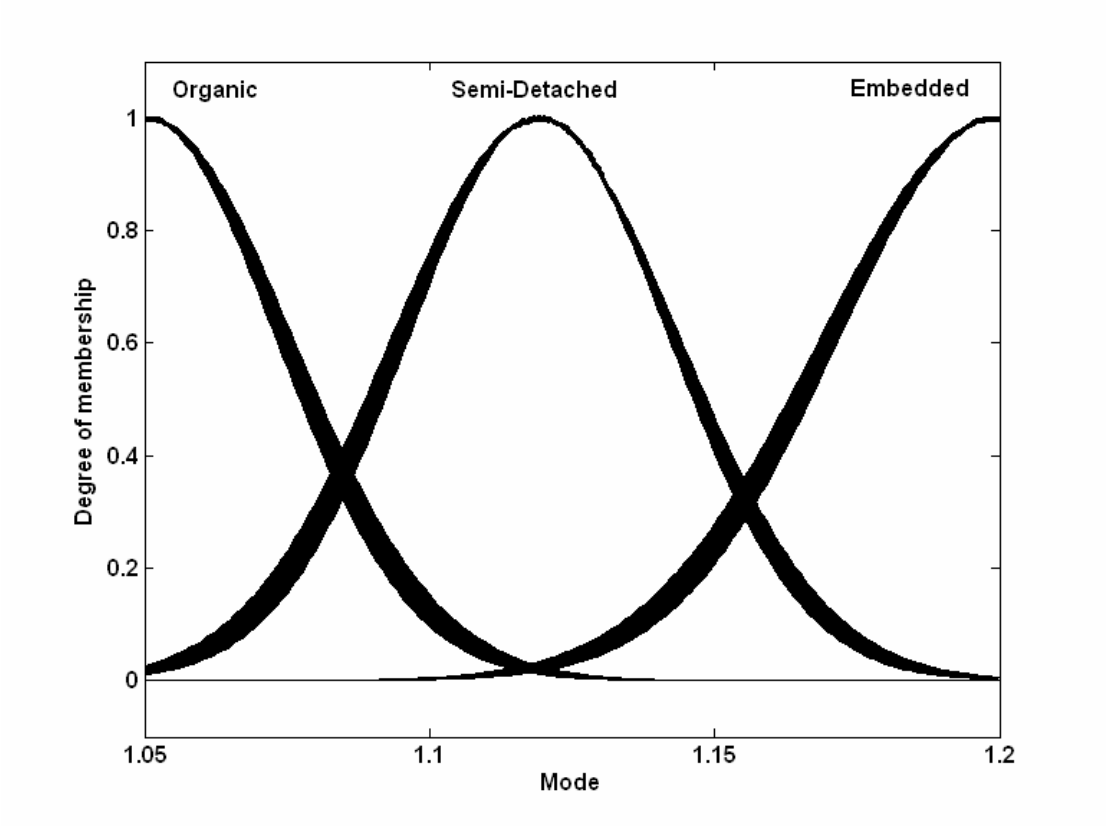

Figure 5-4: Type-2 Gaussian membership functions for mode

[6] Increasing the number of fuzzy sets may result in increasing the prediction accuracy but it will require defining more rules in the rule base and consequently requiring more time for training.

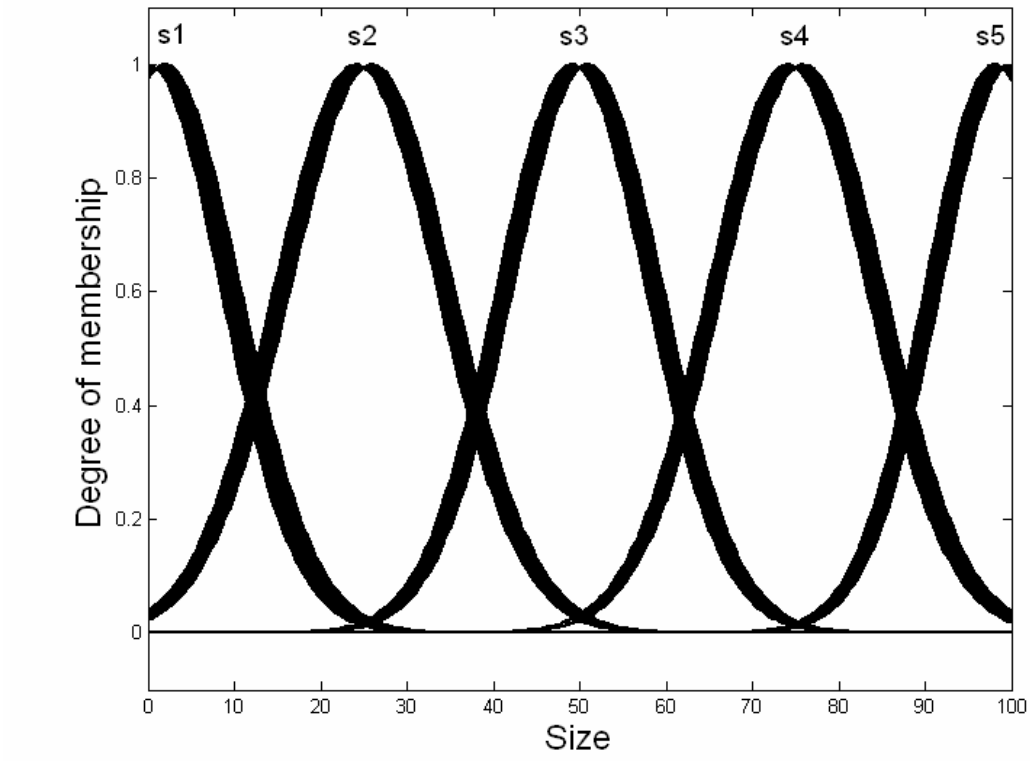

Figure 5-5: Type-2 Gaussian membership functions for size.

With regard to the initialization of the parameters of membership functions i.e. mean (M), standard deviations (σ_A) etc, Rahman [56] has provided a general approach for initializing these parameters which may be altered based on the nature of the problem. Therefore, we have modified the initialization procedure proposed by him where necessary. Let us assume that we need to initialize p different antecedent membership functions. Assume also that we have historical data from n software projects; where there are m different measurements for *mode* and *size*. Table 5-2 represents the structure of the corresponding input dataset where 'Measure' means *mode* or *size*.

Table 5-2: Input dataset structure

Project No.	Measure 1	Measure 2	…	Measure m	Mean of Measures	Standard Deviation of Measures
1	S11	S21	…	Sm1	μ_1	σ_1
2	S12	S22	…	Sm2	μ_2	σ_2
…	…	…	…	…	…	…
N	S1n	S2n	…	Smn	μ_n	σ_n

Now, we calculate the following:

$R^7 = Mean(\sigma_1, \sigma_2, ..., \sigma_n)$

$M1_i{}^8 = M_i - \alpha R$

$M2_i{}^9 = M_i + \alpha R$

$\sigma_{Ai} = \beta R$

Where,

$i = 1 \ldots p$ and

---

[7] It gives an idea of the width of the type-2 membership functions that can be set to handle uncertainty around their means.

[8]

[9] $M1_i$ and $M2_i$ define the width of i^{th} type-2 actecedent MFs.

α and β are real constants and should be defined (using experience or observing the training behavior) to properly encounter the uncertainty around means of MFs and to cover the whole spectrum, respectively. Now we will see how to initialize consequents.

The type-2 FLS derived by Mendel provides control over the type reduced[10] output fuzzy sets (and not the actual type-2 consequents), which are type-1 interval sets. In this framework and the other (for handling laziness/ignorance) discussed later in Section 5.4, the consequents are defined by all the possibilities of the combinations of fuzzy sets in the antecedents. Therefore the position of each consequent interval set is computed by putting the means of the corresponding mode and size fuzzy sets into the COCOMO nominal effort equation. The spread of theses interval sets represents interval of confidence and can be initialized by using past experience or by observing the training behavior.

Finally, we need to initialize the input membership functions. This initialization depends upon the type of inputs we are using: singleton or non-singleton. In the case of singleton inputs the mean value of each input membership function is the corresponding mean of

---

[10] The type-2 FLS, in the first place, produces a type-2 fuzzy set as an output whenever an input is propagated. This type-2 output fuzzy set is then reduced to type-1 interval fuzzy set. This process of reduction of type-2 output fuzzy set to type-1 interval fuzzy set is called as type reduction. The reduced type-1 interval fuzzy set is then used to produce crisp output. Please refer to Chapter 10 of Mendel's book [49] for deep understanding.

the various measures as computed in Table 5-2, e.g., μ_i represent the mean of the i^{th} input membership function. The standard deviation (σ_{sn}) of each input membership function is the corresponding standard deviation of the various measures as computed in Table 5-2:

$$\sigma_{sni} = \sigma_i$$

Where $i = 1 \ldots n$

In the case of non-singleton inputs, the structure of the dataset is the same as represented in Table 5-2 with a little difference that now the various measures are represented by Gaussian MFs. The mean value of each input membership function, in this case, would be the average of means of various measures for a particular project. The standard deviations[11] of each input membership function i.e., σ_{sn1} and σ_{sn2} are computed as:

$$\sigma_{sn1i} = \min(\sigma_{sn1}, \sigma_{sn2}, ..., \sigma_{snm})$$

$$\sigma_{sn2i} = \max(\sigma_{sn1}, \sigma_{sn2}, ..., \sigma_{snm})$$

Where $i = 1 \ldots n$

---

[11] Since the input membership functions, in this thesis, for non-singleton inputs are represented using type-2 Gaussian with uncertain standard deviation, therefore two standard deviation values are required to define the width of the MFs.

5.3.2. Rules Formulation

In the rules formulation we follow the same approach as proposed by Ahmed *et al.* [2], i.e., all the possible combinations of antecedent fuzzy sets are employed. The difference is that in our case we use type-2 fuzzy sets instead of type-1 fuzzy sets. The rules appear as follows:

$$R_1: \textit{IF mode is organic AND size is } \widetilde{s}_1 \textit{ THEN cost is } \widetilde{C}_{11}$$

$$R_2: \textit{IF mode is semi-detached AND size is } \widetilde{s}_1 \textit{ THEN cost is } \widetilde{C}_{12}$$

$$R_3: \textit{IF mode is embedded AND size is } \widetilde{s}_1 \textit{ THEN cost is } \widetilde{C}_{13}$$

$$R_4: \textit{IF mode is organic AND size is } \widetilde{s}_2 \textit{ THEN cost is } \widetilde{C}_{21}$$

$$R_5: \textit{IF mode is semi-detached AND size is } \widetilde{s}_2 \textit{ THEN cost is } \widetilde{C}_{22}$$

$$R_6: \textit{IF mode is embedded AND size is } \widetilde{s}_2 \textit{ THEN cost is } \widetilde{C}_{23}$$

$$\cdots\cdots\cdots$$

In general, we can write this as:

$$R_l: \textit{IF mode is } \widetilde{m}_j \textit{ AND size is } \widetilde{s}_i \textit{ THEN cost is } \widetilde{C}_{ij} \ (1 \le i \le n, 1 \le j \le 3)$$

Where, l is the number of rules equals to $3n$.

As a default we classify size into five fuzzy sets. Therefore, three fuzzy sets of mode and five fuzzy sets of size will produce fifteen rules i.e., fifteen consequent fuzzy sets.

5.3.3. Rules Training

The rules are trained to improve their accuracy in predicting nominal effort. In this thesis, the training is carried out by propagating[12] inputs through FLS and modifying the parameters of various membership functions based on computed error and steepest descent approach, see Algorithm 5-1. In order to train, we need to prepare training dataset from the available dataset. The structure of the available dataset is represented in Table 5-3. The different modes and sizes associated with each project are meant to represent the inputs from the different experts and measurements as discussed earlier.

Table 5-3: Structure of the available dataset

Project No.	Mode 1	Mode 2	…	Mode m	Size 1	Size 2	…	Size m	Effort
1	M11	M21	…	Mm1	S11	S21	…	Sm1	E1
2	M12	M22	…	Mm2	S12	S22	…	Sm2	E2
…	…	…	…	…	...	...	…	…	…
n	M1n	M2n	…	Mmn	S1n	S2n	…	Smn	En

We extract the training dataset as depicted in Table 5-4. Each row in Table 5-3 provides a data point as depicted in Table 5-4. The values of effort are the same as provided in the available dataset, whereas for each data point the definition of input membership functions for mode (MF_{mi}) and size (MF_{si}) follows the approach as discussed in Section

[12] Fuzzifying inputs with the antecedent membership functions and producing the output.

5.3.1, see Figure 5-6 and 5-7. In our experiments, the training dataset is formed by selecting k data points that is approximately 70 percent of the overall available dataset.

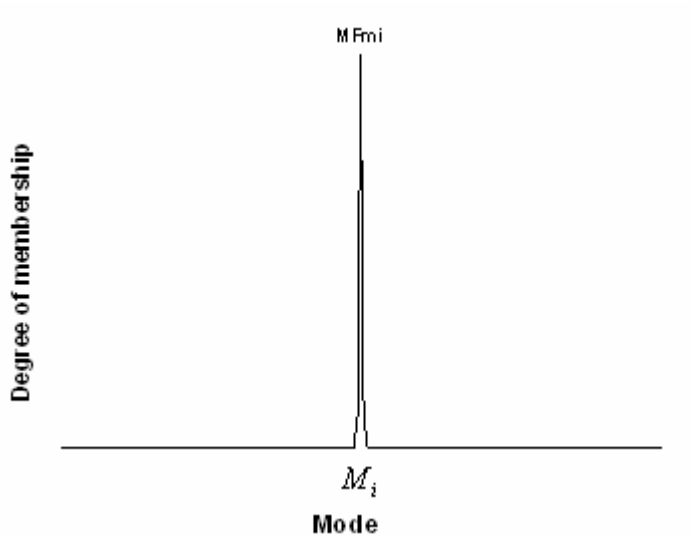

Figure 5-6: An example of input membership function[13] for mode

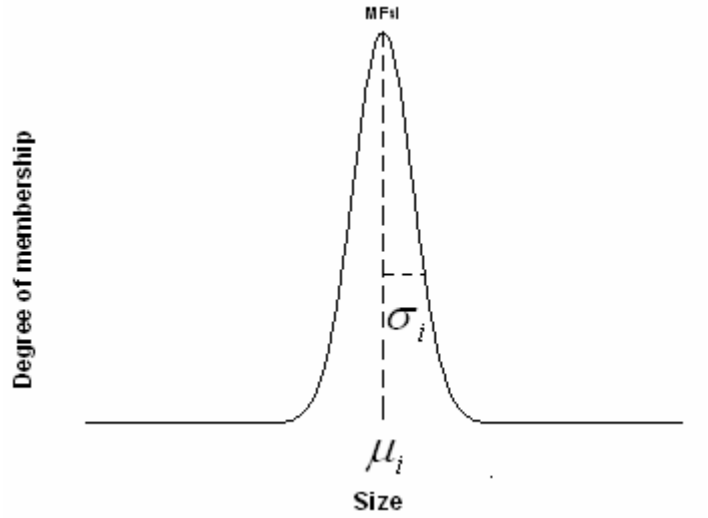

Figure 5-7: An example of input membership function for size

[13] It is narrow as compared to input membership function for size, because mode spans over a very short interval [1.05, 1.2] as compared to size.

Table 5-4: Training dataset structure

Mode	Size	Effort
MF_{m1}	MF_{s1}	E1
MF_{m2}	MF_{s2}	E2
…	…	…
MF_{mk}	MF_{sk}	Ek

Algorithm 5-1: Training algorithm for tuning type-2 FLS

Given N input-output training samples $(x^{(t)} : y^{(t)})$, $t=1,..., N$. Objective is to minimize the error function for K training Epochs. The error function is computed as:

$$E(t) = \frac{1}{2}\left(f(x^{(t)}) - y^{(t)}\right)^2 \quad t = 1 \ ..., N$$

Steps

1. Initialize all the parameters.
2. Set the counter, e, of the training epoch to zero i.e. e=0.
3. Set the counter, t, of the training data to one. i.e., t=1.
4. Apply the means of mode and size with their corresponding standard deviation to the non-singleton type-2 FLS and compute the output $f(x^{(i)})$.
5. Compute the output error (relative) as: $e = \frac{f(x^{(i)}) - y^{(i)}}{y^{(i)}}$
6. Tune the uncertain means of the antecedent membership functions and the intervals of consequents using steepest descent algorithm for the error function.
7. Set t=t+1. If t = N+1, go to next step otherwise apply the next input.
8. Set e=e+1. If e=K, Stop; otherwise start a new epoch.

We have used the training algorithm proposed by Rahman [56] as described in Algorithm 5-1.

5.3.4. Framework Validation

The accuracy of the trained rules is assessed through validation. In order to validate, we need to prepare validation dataset from the available dataset. We extract validation dataset from over all dataset as represented in Table 5-5. In this dataset we extract m x k data points which is approximately 30 percent of the over all dataset.

Table 5-5: Structure of the validation dataset

Mode	Size	Effort
MF_{m11}	MF_{s11}	E1
MF_{m21}	MF_{s21}	E1
…	…	…
MF_{mm1}	MF_{sm1}	E1
MF_{m12}	MF_{s12}	E2
MF_{m22}	MF_{s22}	E2
…	…	…
MF_{mm2}	MF_{sm2}	E2
…	…	…
MF_{m1k}	MF_{s1k}	Ek
MF_{m2k}	MF_{s2k}	Ek
…	…	…
MF_{mm2}	MF_{smk}	Ek

The validation dataset structure is similar to the training dataset structure. The only difference here is that we use original values of measures, and not their average, as the mean of input MFs. The reason for this is that during operations of the prediction system, software engineers do not consider the uncertainty in their measured values for the internal attribute because most often each engineer uses only a single metric, which they think is the most reliable. The point is that the measurements are typically used with their inherent noise.

The non-singleton fuzzification is used during validation. In order to do so, we need to compute standard deviation for input membership functions. In the case of non-singleton inputs, standard deviation associated with each input is assigned as the standard deviation of input MFs, whereas singleton inputs do not possess standard deviation therefore some suitable value is assigned as the standard deviation.

It is worth noting here that we adopt an approach for validation that is slightly different and much better than that proposed by Rahman [56]. Rahman has proposed to use singleton fuzzification for this purpose. We suggest the use of non-singleton fuzzification, which results in increased accuracy of the prediction system. The reason for this is that the rules are trained by using non-singleton fuzzification. If we apply singleton inputs during validation then it means we are not providing useful information which is standard deviation (observed by the prediction system during training) of the input MFs. Thus this incomplete information, for validation, will result in reduced accuracy of the prediction system.

5.4. Uncertainty Handling Due to Laziness/Ignorance in Effort Prediction Framework

We have already discussed in Chapter 4 that the FLS based effort prediction framework proposed by Liang and Noore[73] has introduced laziness/ignorance. In order to accommodate the resulting uncertainty they thought that type-1 FLS would be a better option, but according to Mendel such type of uncertainty can only be handled using type-2 FLS [49]. Their framework is discussed in Chapter 4. We have followed a slightly different approach than Liang and Noore to construct our framework as evident from Figure 5-8. This multistage framework is similar[14] to the framework presented in Figure 5-3 with a difference that here we have included the details that are proposed to handle uncertainty that arises due to considering only six cost drivers (RELY, CPLX, TIME, ACAP, PCAP and PCON) information, as suggested by Liang and Noore. In what follows, we will provide the details as how to initialize, train and activate the different stages of the effort prediction framework.

---

[14] The nominal effort prediction stage is the same as we have discussed in Section 5.3. Moreover, if all the cost drivers information is available then the details provided (for the first stage) in this section can be applied to each cost driver to produce corresponding effort multiplier. The effort multipliers can then be multiplied to produce EAF.

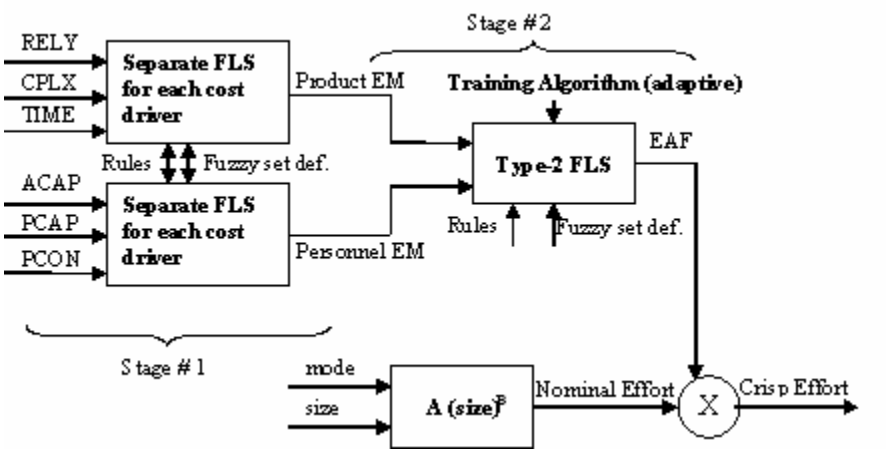

Figure 5-8: Type-2 FLS based framework for handling laziness/ignorance in cost prediction

5.4.1. First Stage

In the first stage we have a separate fuzzy logic system for each input cost driver. This is because it provides an ease in relating each cost driver with the corresponding effort multiplier. The outputs (effort multipliers) from these FLSs are divided into two groups, one to compute product effort multiplier and the other to compute personnel effort multiplier. The product effort multiplier is calculated by multiplying the effort multiplier values of RELY, CPLX and TIME; whereas personnel effort multiplier is calculated by multiplying the effort multiplier values of ACAP, PCAP and PCON. The values of the cost drivers are usually provided by experts and we have already made this clear in the previous discussion that different experts may provide different values for a particular input. If this is the case then we will follow the similar approach as we have explained in the context of uncertainty handling in Section 5.3. But in this framework we are concerned with uncertainty that arises due to laziness or ignorance, that's why the

approach here is quite different. Therefore, we assume that only a single expert is asked to provide the input values, this way we do not assume any uncertainty in the inputs for the first stage. Assuming the scenario we can employ non-adaptable type-1 FLS for this stage. Now, we will discuss how to initialize, formulate and activate FLSs.

Initializing the Framework

The antecedent and consequent classification into fuzzy regions for RELY in the first stage is shown in Figure 5-9 and 5-10, respectively. All the other cost drivers can be treated in the same manner.

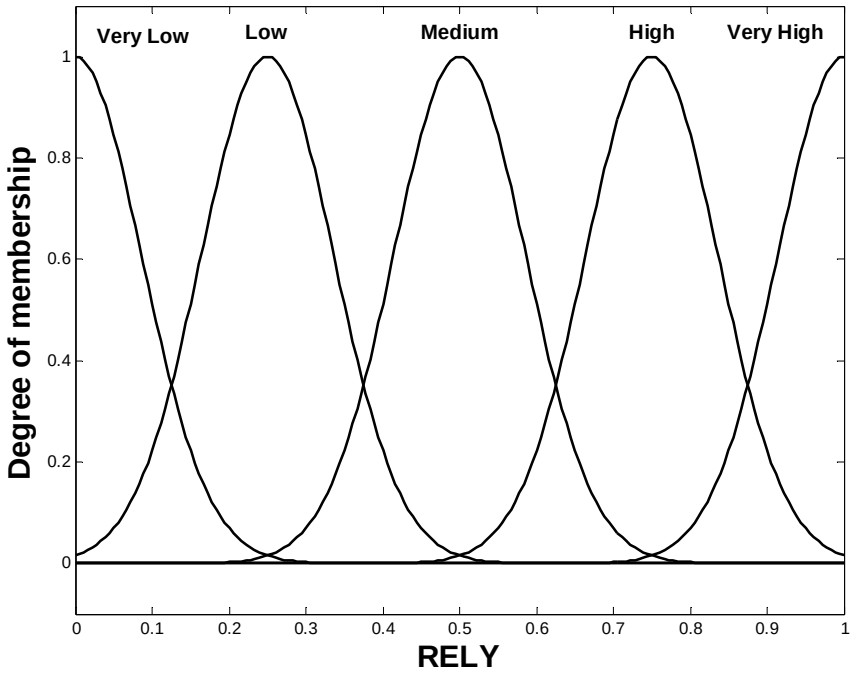

Figure 5-9: Classification of RELY antecedent into fuzzy regions

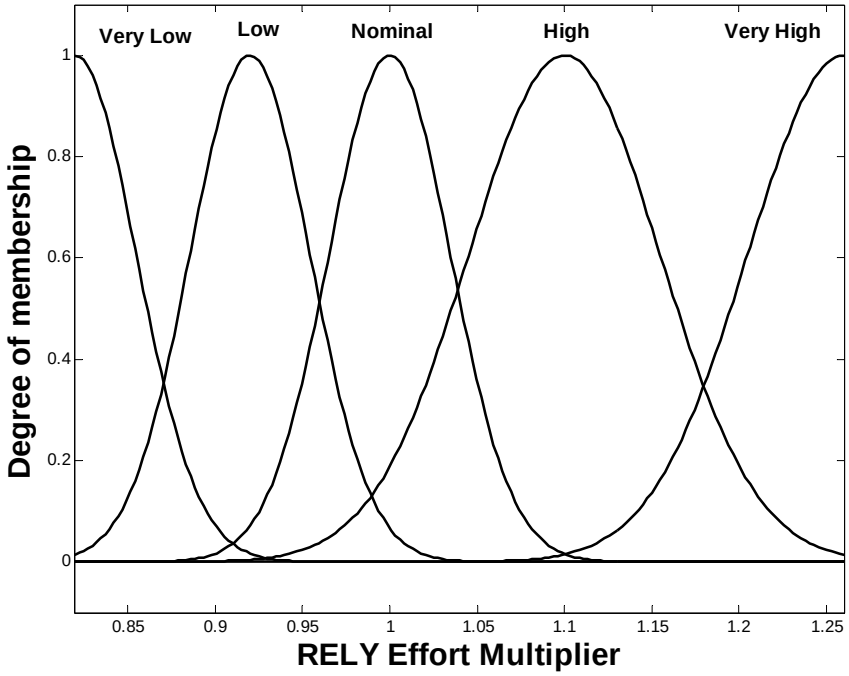

Figure 5-10: Classification of RELY consequent into fuzzy regions

Rules Formulation

The rules for FLSs are of the form.

$$R_1 : IF\ cost\ driver\ X\ is\ low\ THEN\ EM_X\ is\ low$$

$$R_2 : IF\ cost\ driver\ X\ is\ medium\ THEN\ EM_X\ is\ medium$$

…

In general, we can write this as:

$$R_l : IF\ cost\ driver\ X\ is\ FS\ THEN\ EM_X\ is\ FS$$

Where

X is any cost driver

EM_X is the corresponding effort multiplier

FS is any fuzzy set and

l is the number of rules

Activating the FLSs of the First Stage

Since we assume no uncertainty in the inputs and outputs of the first stage FLSs, that's why inputs are fuzzified using type-1 singleton fuzzification and the corresponding output are calculated.

5.4.2. Second Stage

This stage is composed of a single FLS with product and personnel effort multipliers as the inputs and effort adjustment factor (EAF) as the output. This stage introduces uncertainty in the EAF because it originally depends on 17 cost drivers, whereas here we have only two inputs that are computed using only 6 cost drivers. Therefore we propose to use type-2 FLS in this stage instead of type-1 FLS. In the following, we will discuss how to initialize, formulate, train and activate this stage.

Initializing the Framework

The antecedents for this stage are the product and the personnel effort multipliers while the EAF is the consequent. We consider our antecedents and consequent membership functions to be type-2 Gaussian with uncertain mean. Since there is no uncertainty due to noise in the outputs (inputs of this stage) of the first stage therefore we use type-2 singleton fuzzification.

After defining type of the membership functions we need to classify antecedents and consequents into suitable number of fuzzy sets. We have divided the antecedents into five fuzzy sets each.

The consequents are set by simply multiplying the mean of each MF of one antecedent with all the means of the other antecedent MFs.

Now, we need to initialize the parameters of membership functions i.e. means, standard deviations etc. The parameters of the antecedents can be initialized by running experiments and looking at the behavior of training because, as discussed earlier, here we assume uncertainty due to laziness that's why we cannot compute means and standard deviations by following the approach discussed in Section 5.4. The uncertainty is seen in the outputs due to not including all the cost drivers. Therefore, one can initialize the consequents by analyzing the training data or by running experiments and noticing the behavior of training.

Rules Formulation

We consider all the possibilities of relationship between the two antecedents therefore we have altogether twenty five rules in the FLS of the second stage. The rules for FLSs are of the form:

R_1 *: IF product EM is very low and personnel EM is very low THEN EAF is very low*

R_2 *: IF product EM is very low and personnel EM is low THEN EAF is low*

R_3 *: IF product EM is medium and personnel EM is medium THEN EAF is medium*

…

In general we can write this as:

$$R_l : \textit{IF product EM is } \widetilde{p}_i \textit{ and personnel EM is } \widetilde{q}_j \textit{ THEN EAF is } \widetilde{G}_{ij} \; (1 \le i \le 5, 1 \le j \le 5)$$

Where, l is the number of rules equals to 25.

Rules training

We apply an algorithm similar to the one discussed in Section 5.3.3. The only difference is that here we use singleton fuzzification because each input has a single value (crisp value) for a particular project. The non-singleton fuzzification can not be applied here because it requires defining the input MFs, which in turn require computation of standard deviations. The standard deviation information can only be acquired when more than one value is provided for each input for a particular project.

Validation and Activation of the Second Stage

We apply singleton fuzzification to compute the effort adjustment factor (EAF). Finally EAF can be multiplied with the nominal effort to produce the crisp effort.

CHAPTER 6

IMPACT OF ALGORITHMS, PARAMETERS AND ARCHITECTURE ON THE PERFORMANCE OF FLS-BASED EFFORT PREDICTION FRAMEWORK

The nature of training algorithm, parameters and architecture[15] play a vital role in a FLS[16] based effort prediction model. The various building blocks of a FLS are shown in Figure 2-2. These building blocks have various parameters; refer to Figure 6-1, which can be set in different ways in order to build a FLS. In addition, a FLS can be trained or tuned to optimize developed fuzzy rules using different algorithms e.g., steepest descent approach

---

[15] Architecture describes how various components of framework are combined to produce a prediction system.

[16] The reader is advised to refer to the first three chapters of Wang's book [76] for better understanding of the various Fuzzy Logic terms and parameters (although they are briefly discussed in Chapter 2) of a Fuzzy Logic System.

[76] and heuristic based approach [53]. Our aim is to investigate impact of various combinations and nature of training algorithms in an effort to find out the one that can provide better results in the context of software development effort prediction models. Moreover, the components of COCOMO and FLS based cost prediction frameworks can be integrated using different architectures. Therefore we also like to study architectural impact on effort prediction framework. In order to carry out aforementioned investigations, we need to develop the corresponding FLSs. Therefore in what follows, we will discuss how different FLSs are developed for this task.

6.1. Fuzzy Inference

In the following type-1 FLS based effort prediction models we have employed Mamdani's[17] minimum implication [76][82] and Larsen's product implication[18] [44] while developing fuzzy inference engine. In this respect the heuristic based approach uses max-min inference, whereas the steepest descent approach utilizes max-product implication.

The reason for this is the fact that minimum and product implications are the most widely used implications in the today's engineering applications of fuzzy logic. They are

---

17

18 Mamdani and Larsen defined implication methods that can be used to derive FLSs. Examples are the Wang's[76] and Mendel's[49] type-1 and type-2 FLSs, respectively. Also in this thesis, these implication methods are used to derive the desired FLSs.

referred to as *Engineering Implications*. Actually these implications provide simpler solutions to the problems associated with other implications. Although other implications agree with the accepted propositional logic definition of implication but, if applied, the result of firing a specific rule, whose consequent is associated with a specific fuzzy set of finite support, is a fuzzy set whose support is infinite. Moreover, their firing result is also affected by some bias [49].

6.2. Types of Training Algorithms

In this thesis, we have used the back propagation algorithm in the context of adapting a FLS [49]. In the back propagation algorithm none of the antecedents or consequents parameters are fixed ahead of time. They can be all tuned using the following two most common approaches in the FLS community.

i. Steepest Descent Approach and

ii. Heuristic Based Approach

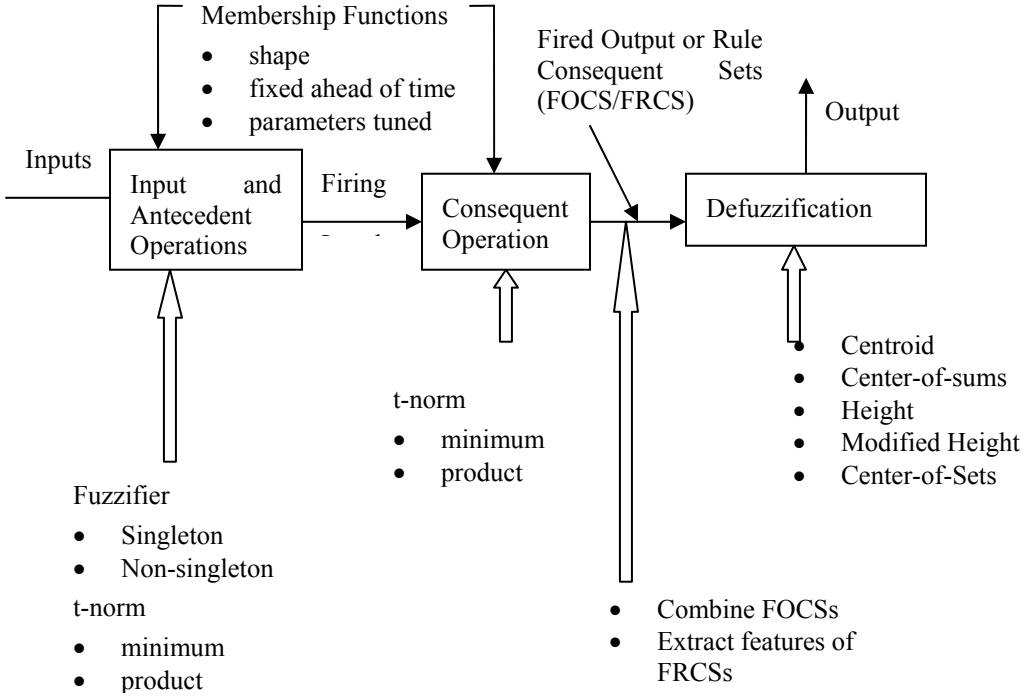

Figure 6-1: Choices that need to be made to design a type-1 FLS [49]

6.2.1. Steepest Descent Approach

Consider a FLS with singleton fuzzification, max-product composition, product implication, height defuzzification and Gaussian membership functions. It is given by the equation:

$$y(x^{(i)}) = f_s(x^{(i)}) = \frac{\sum_{l=1}^{M} \bar{y}^{l} \prod_{k=1}^{p} \exp\left[-\frac{\left(x_k^{(i)} - m_{F_k^l}\right)^2}{\left(2\sigma_{F_k^l}^2\right)}\right]}{\sum_{l=1}^{M} \prod_{k=1}^{p} \exp\left[-\frac{\left(x_k^{(i)} - m_{F_k^l}\right)^2}{\left(2\sigma_{F_k^l}^2\right)}\right]} \qquad i = 1,\ldots,N \qquad (6\text{-}1)$$

Where

M is number of rules,

p is number of antecedents and

N is number of data points

Given an input-output training pair $(x^{(i)} : y^{(i)})$ also known as data point, we wish to design FLS in (5-1) such that the following error function is minimized:

$$e^{(i)} = \frac{1}{2}\left[f_s(x^{(i)}) - y^{(i)}\right]^2 \qquad i = 1,...,N \tag{6-2}$$

It can be seen from (6-1) that f_s is completely characterized by $\bar{y}^l$ (point at which rule l has the highest degree of membership), $m_{F_k^l}$ (mean of k^{th} antecedent in rule l) and $\sigma_{F_k^l}$ (standard deviation of k^{th} antecedent in rule l). Steepest descent approach can be applied to obtain the following recursions to update all the design parameters of this FLS in order to minimize error function in (6-2) ($k = 1,..., p$, $l = 1,..., M$ and $i = 0,1,...$):

$$m_{F_k^l}(i+1) = m_{F_k^l}(i) - \alpha_m[f_s(x^{(i)}) - y^{(i)}][\bar{y}^l(i) - f_s(x^{(i)})] \times \frac{[x_k^{(i)} - m_{F_k^l}(i)]}{\sigma^2_{F_k^l}(i)} \phi_l(x^{(i)}) \tag{6-3}$$

$$\bar{y}^l(i+1) = \bar{y}^l(i) - \alpha_{\bar{y}}[f_s(x^{(i)}) - y^{(i)}]\phi_l(x^{(i)}) \tag{6-4}$$

and

$$\sigma_{F_k^l}(i+1) = \sigma_{F_k^l}(i) - \alpha_\sigma[f_s(x^{(i)}) - y^{(i)}][\bar{y}^l(i) - f_s(x^{(i)})] \times \frac{[x_k^{(i)} - m_{F_k^l}(i)]^2}{\sigma^3_{F_k^l}(i)} \phi_l(x^{(i)}) \tag{6-5}$$

Note: The definitions of all the symbols and variables described in this section are applicable to the rest of the chapter unless otherwise stated.

Now, the back propagation algorithm can be applied as described in Algorithm 6-1.

Algorithm 6-1: Back propagation algorithm for type-1 FLS

1. Initialize the parameters of all the membership functions for all the rules, $m_{F_k^l}(0)$, $\bar{y}^l(0)$ and $\sigma_{F_k^l}(0)$.
2. Choose the learning parameters, α_m, $\alpha_{\bar{y}}$ and α_σ. Usually they are chosen to be the same, say α.
3. Set some end criterion to achieve convergence.
4. **Repeat**
   i. **for all** data points $(x^{(i)} : y^{(i)})\, i = 1,..., N$
      a) Propagate the next data point through the FLS and compute $f_s(x^{(i)})$.
      b) Compute error as: $e = f_s(x^{(i)}) - y^{(i)}$
      c) Update the parameters of the membership functions using (6-3), (6-4) and (6-5).
   ii. **end for** *(*end for each input-output pair*)*
   iii. Compute the root mean square relative error (RMSRE) as (6-6).
   iv. Test the end criterion. If satisfied break.

Until *(*end for each epoch*)*

$$\text{RMSRE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left[\frac{f_s(x^{(i)}) - y^{(i)}}{y^{(i)}}\right]^2} \tag{6-6}$$

6.2.2. Heuristic Based Approach

A detailed description of the heuristic based algorithm that is studied in this thesis can be found in [53]. We only provide a brief account on how this approach works. Like the steepest descent approach, in each loop the algorithm propagates a data point, determines the output of FLS and computes the output error. The main information derived from the error value is whether the contribution of a fuzzy rule to the overall output values should be increased or decreased [53]. The parameter updates of antecedents[19] and consequent membership functions (MFs) are then determined using the error value based on some heuristics.

The consequent of FLS is modified using heuristic that takes defuzzification procedure into consideration. The aim is to move the output of FLS closer to the target value. This is achieved by shifting the support of the consequent fuzzy set such that the center of fuzzy set moves closer to the target value. The support of a fuzzy set (to be modified) is reduced to focus the fuzzy set on the target value or extended towards the target value if the target value has non zero or zero membership with the fuzzy set, respectively. Figure 6-2 gives an example of modification of triangular membership function in the consequent [53].

---

[19] Refer to the work by Nauck [53] for details regarding modification of antecedent membership functions.

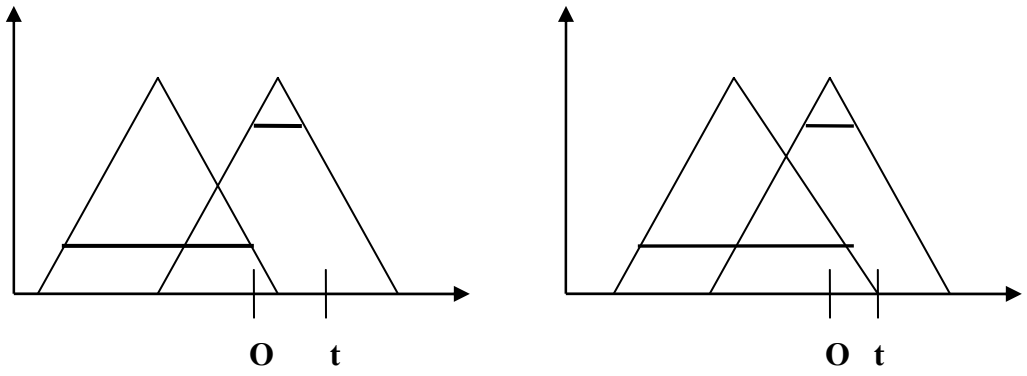

Figure 6-2: To move an output value *O* of a fuzzy system closer to a current target value t the consequent fuzzy set is move towards t [53].

The important point here to note is that in the case of steepest descent approach usually all the rules are modified on the basis of computed error, whenever a data point is propagated through the FLS. Moreover, different rules do not share the same MFs, i.e., the instances of a MF of a particular fuzzy set across various rules are independent of each other. For example, the instances of MF belongs to fuzzy set LOW of an antecedent, say *size*, that appear in more than one rule may have different parameters values. But in the case of heuristic based approach various rules share the same instance of a particular fuzzy set. Finally, the rule which contributes to the output value is modified.

6.3. Adaptive Fuzzy Logic System Parameters

The parameters that can affect the behavior of FLS are shown in Figure 6-1 above. We have investigated the impact of the following on the accuracy of the software development effort prediction systems generated using the framework:

i. Height defuzzification versus modified height defuzzification.

ii. Triangular membership functions versus Gaussian membership functions.

iii. Different ways of processing the error before back propagating it: a) Normalized error and b) Relative error.

6.3.1. Height Defuzzifier

The height defuzzifier is also known as the center average defuzzifier. In this defuzzifier each rule output fuzzy set is replaced by a singleton at the point having maximum membership in that output set. The centroid of the type-1 set, comprised of these singletons, is then calculated. The output of a height defuzzifier is given as:

$$y_h(x) = \frac{\sum_{l=1}^{M} \bar{y}^l \mu_{B^l}(\bar{y}^l)}{\sum_{l=1}^{M} \mu_{B^l}(\bar{y}^l)} \tag{6-7}$$

Where $\bar{y}^l$ is the point having maximum membership in the l^{th} output set, and its membership grade in the l^{th} output set is $\mu_{B^l}(\bar{y}^l)$. The resulting FLS with height defuzzifier together with parameter update equations is described in Section 6.1.1.

The problem with height defuzzifier is that it only uses the center of the support, $\bar{y}^l$, of the consequent membership function. It does not take into consideration the shape of the consequent membership function whether the membership function is narrow; which can be interpreted as an indication of a very strong belief in that rule, or is broad, which can

be interpreted as an indication of much less belief in that rule, the height defuzzifier produces the same result [49].

6.3.2. Modified Height Defuzzification

The modified height defuzzifier takes care of the deficiency of height defuzzifier. It is very similar to the height defuzzifier; the only difference is that in the modified height defuzzifier each $\mu_{B^l}(\bar{y}^l)$ is scaled by the inverse of the spread of the l^{th} consequent set. The output is given as:

$$y_h(x) = \frac{\sum_{l=1}^{M} \bar{y}^l \mu_{B^l}(\bar{y}^l) \Big/ \delta^l}{\sum_{l=1}^{M} \mu_{B^l}(\bar{y}^l) \Big/ \delta^l} \tag{6-8}$$

Where δ^l is some measure of the spread of the l^{th} consequent set, $\bar{y}^l$ and $\mu_{B^l}(\bar{y}^l)$ have the same meaning as in (6-7). For triangular or trapezoidal membership functions, δ^l could be the support of these functions, whereas for Gaussian membership functions, δ^l could be its standard deviation.

Now, consider a FLS with singleton fuzzification, max-product composition, product implication, modified height defuzzification and Gaussian membership functions. It is represented as:

$$y(x^{(i)}) = f_s(x^{(i)}) = \frac{\sum_{l=1}^{M} \bar{y}^{l} \prod_{k=1}^{p} \exp\left[-\frac{\left(x_k^{(i)} - m_{F_k^l}\right)^2}{\left(2\sigma_{F_k^l}^2\right)}\right] \Big/ \delta^l}{\sum_{l=1}^{M} \prod_{k=1}^{p} \exp\left[-\frac{\left(x_k^{(i)} - m_{F_k^l}\right)^2}{\left(2\sigma_{F_k^l}^2\right)}\right] \Big/ \delta^l} \quad i = 1,...,N \qquad (6\text{-}9)$$

Steepest descent approach can be applied to obtain the following recursions to update all the design parameters of this FLS in order to minimize error function in (6-2) ($k = 1,...,p$, $l = 1,...,M$ and $i = 0,1,...$):

$$m_{F_k^l}(i+1) = m_{F_k^l}(i) - 2\alpha_m \left(\frac{f_s(x^{(i)}) - y^{(i)}}{\sum_{l=1}^{M} z^l / \delta^l} \right) [\bar{y}^{l}(i) - f_s(x^{(i)})] \times \frac{[x_k^{(i)} - m_{F_k^l}(i)]}{\sigma_{F_k^l}^2(i)} \frac{z^l}{\delta^l} \qquad (6\text{-}10)$$

$$\bar{y}^{l}(i+1) = \bar{y}^{l}(i) - \alpha_{\bar{y}} [f_s(x^{(i)}) - y^{(i)}] \frac{z^l / \delta^l}{\sum_{l=1}^{M} z^l / \delta^l} \qquad (6\text{-}11)$$

and

$$\sigma_{F_k^l}(i+1) = \sigma_{F_k^l}(i) - 2\alpha_\sigma \left(\frac{f_s(x^{(i)}) - y^{(i)}}{\sum_{l=1}^{M} z^l / \delta^l} \right) [\bar{y}^{l}(i) - f_s(x^{(i)})] \times \frac{[x_k^{(i)} - m_{F_k^l}(i)]^2}{\sigma_{F_k^l}^3(i)} \frac{z^l}{\delta^l} \qquad (6\text{-}12)$$

$$\sigma^l(i+1) = \delta^l(i) + \alpha_\delta [f_s(x^{(i)}) - y^{(i)}][\bar{y}^{l}(i) - f_s(x^{(i)})] \frac{z^l}{\delta^{l2}(i)} \left(\frac{1}{\sum_{l=1}^{M} z^l / \delta^l} \right) \qquad (6\text{-}13)$$

Where

$$z^l = \prod_{k=1}^{p} \exp\left[-\frac{\left(x_k^{(i)} - m_{F_k^l}\right)^2}{\left(2\sigma_{F_k^l}^2\right)} \right] \quad (6\text{-}14)$$

Now, the back propagation algorithm can be applied in a fashion similar to that explained in Section 6.1.1.

6.3.3. Gaussian and Triangular Membership Functions

Besides other MFs, Gaussian and triangular MFs, see Figure 6-3, can be used to define antecedents or consequent fuzzy sets. In this study we have investigated the impact of these MFs on the software development cost estimation. An adaptive FLS with triangular MFs and heuristic based approach for cost estimation is already investigated by Saliu [63]. In the following we will develop an adaptive FLS with triangular MFs using steepest descent approach.

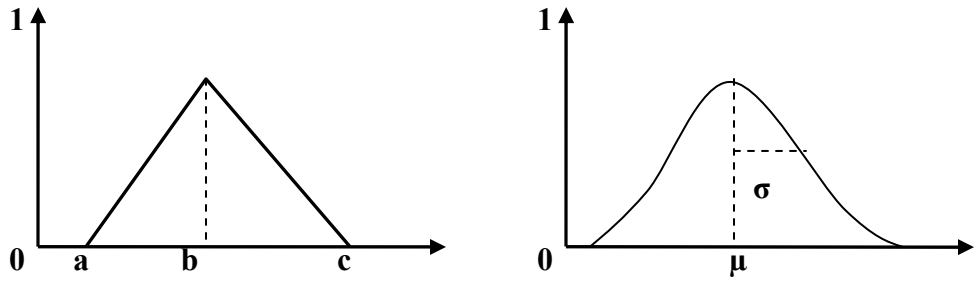

Figure 6-3: Gaussian and triangular MFs

A triangular MF is a three point function, defined by minimum (a), maximum (c) and modal (b) values i.e., TMF(a,b,c), where ($a \le b \le c$). The membership value at any input x can be calculated as follows:

$$\mu_{a,b,c}(x) = \begin{cases} \dfrac{x - a}{b - a} & a \le x \le b \\ \dfrac{c - x}{c - b} & b \le x \le c \\ 0 & otherwisw \end{cases} \tag{6-15}$$

Now, consider a FLS with singleton fuzzification, max-product composition, product implication, height defuzzification and triangular MFs. It is given by the equation:

$$y(x^{(i)}) = f_s(x^{(i)}) = \frac{\sum_{l=1}^{M} \bar{y}^l \left(\prod_{k=1}^{p} \left(\frac{x_k^{(i)} - a_{F_k^l}}{b_{F_k^l} - a_{F_k^l}} \right) \right)}{\sum_{l=1}^{M} \left(\prod_{k=1}^{p} \left(\frac{x_k^{(i)} - a_{F_k^l}}{b_{F_k^l} - a_{F_k^l}} \right) \right)} \tag{6-16}$$

Steepest descent approach can be applied to obtain the following recursions to update all the design parameters of this FLS in order to minimize error function in (6-2) ($k = 1,...,p$, $l = 1,...,M$ and $i = 0,1,...$):

When a ≤ x ≤ b

$$a_{F_k^l}(i+1) = a_{F_k^l}(i) - \alpha \frac{[f_s(x^{(i)}) - y^{(i)}]}{b_{F_k^l}(i)} \frac{z^l}{b_{F_k^l}(i) - a_{F_k^l}(i)} \left(1 - \frac{b_{F_k^l}(i) - a_{F_k^l}(i)}{x_k^{(i)} - a_{F_k^l}(i)} \right) [\bar{y}^l(i) - f_s(x^{(i)})] \tag{6-17}$$

$$b_{F_k^l}(i+1) = b_{F_k^l}(i) + \alpha \frac{[f_s(x^{(i)}) - y^{(i)}]}{b_{F_k^l}(i)} z^l \left(\frac{1}{b_{F_k^l}(i) - a_{F_k^l}(i)} \right) [\bar{y}^l(i) - f_s(x^{(i)})] \tag{6-18}$$

When b ≤ x ≤ c

$$c_{F_k^l}(i+1) = c_{F_k^l}(i) - \alpha \frac{[f_s(x^{(i)}) - y^{(i)}]}{b_{F_k^l}(i)} \frac{z^l}{c_{F_k^l}(i) - b_{F_k^l}(i)} \left(\frac{c_{F_k^l}(i) - b_{F_k^l}(i)}{c_{F_k^l}(i) - x_k^{(i)}} - 1 \right) [\bar{y}^l(i) - f_s(x^{(i)})] \tag{6-19}$$

$$b_{F_k^l}(i+1) = b_{F_k^l}(i) - \alpha \frac{[f_s(x^{(i)}) - y^{(i)}]}{b_{F_k^l}(i)} \frac{z^l}{c_{F_k^l} - b_{F_k^l}(i)} [\bar{y}^l(i) - f_s(x^{(i)})] \quad (6\text{-}20)$$

Where

$$z^l = \prod_{k=1}^{p} \frac{\left(x_k^{(i)} - a_{F_k^l}\right)}{\left(b_{F_k^l} - a_{F_k^l}\right)} \quad (6\text{-}21)$$

Finally, the back propagation algorithm can be applied in the similar fashion as explained in Section 6.1.1. The only difference is that during modifying the parameters we must make sure that the condition ($a \le b \le c$) holds.

6.3.4. Normalized and Relative Error

It seems to be a good practice to compute the normalized or relative error for back propagation because this tolerates the small errors and makes the error relative to the output value [63]. In fact it is necessary because training the fuzzy sets for cost estimation takes the ranges of the individual variables into account. Since the parameters updates depend upon the value of the output error and the output (effort) is exponentially related to the inputs therefore the output error is equally so large. Hence in order to be completely independent from the ranges of all the variables, normalized or relative error should be used in the back propagation algorithm [63]. Moreover, because of this there would be no need for normalizing the dataset between [0 1]. The normalized or relative error for output effort can be computed as:

$$\text{Normalized error} = \frac{\text{exp}\,ected\ effort - actual\ effort}{range\ of\ output\ effort\ \text{var}\,iable} \quad (6\text{-}22)$$

$$\text{Relative error} = \frac{\text{exp}\,ected\ \text{effort} - actual\ effort}{\text{exp}\,ected\ effort} \quad (6\text{-}23)$$

6.4. Architectures for FLS Based Software Effort Prediction Framework

Ahmed *et al.* [2] and Liang and Noore [73] FLS based frameworks for effort prediction are presented in Chapter 4. These frameworks have uniquely combined the various components of COCOMO based cost estimation model to produce their architectures. There are two major stages in both the architecture:

i. Nominal effort

ii. Effort adjustment factor (EAF)

Ahmed *et al.* have provided the concepts of fuzziness in both the stages and described how the fuzzy sets can be defined to produce FLSs. The outputs of both the stages are simply multiplied to estimate the software development effort. But he has only provided the experimental results of the nominal effort stage due to unavailability of the experts to provide data for the cost drivers.

Liang and Noore have only fuzzified the EAF stage by considering only 6 cost drivers and they simply multiply the output of this stage with the nominal effort to produce effort value. The EAF stage is further divided into several FLS sub-stages. Each sub-stage represents a unique attribute category e.g., personnel category. Each sub-stage has a number of cost drivers as inputs corresponding to that category. As mentioned earlier this framework introduces uncertainty due to not considering remaining 11 cost drivers.

Although breaking into two stages have their own benefits when one wants to compute only the nominal effort. In this thesis, we have investigated whether combining both the stages can lead to better effort estimates. The proposed architecture is presented next.

6.4.1. Proposed architecture

The proposed architecture is shown in Figure 6-4. In this architecture we first fuzzify the cost drivers to get corresponding effort multipliers and then combine mode, size and computed EAF into a single FLS.

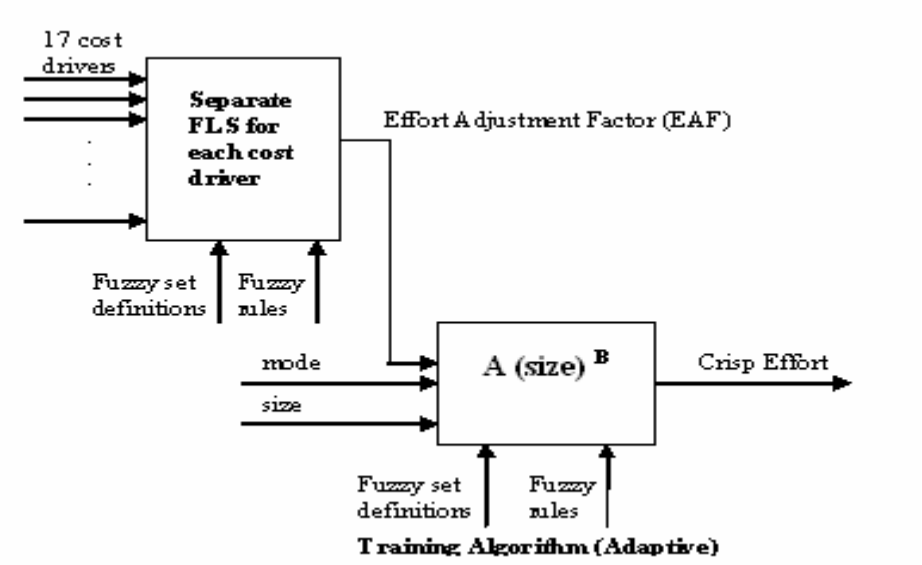

Figure 6-4: Proposed architecture that combines all the components of COCOMO in a single FLS

In the framework cost drivers are fuzzified in the similar way as we have discussed in Section 5.4.1. The difference is that just this time we fuzzify all the 17 cost drivers. The output effort multipliers of these 17 FLSs are multiplied to produce EAF. The computed EAF is then combined with mode and size in the other FLS to estimate effort.

One other possibility could be to combine various categories of effort multipliers first e.g., product or personnel categories etc, and then feed them in the FLS with mode and size. But this will result in much complicated inference engine and consequently will take more time to train and activate.

The antecedents for mode and size are classified into fuzzy sets in the same way as proposed by Ahmed *et al.* [2], whereas the EAF is divided into five equal regions. Gaussian membership functions are used. Rules are formulated by combining all the possibilities of antecedent fuzzy sets i.e., there are 75 rules. The consequents are defined by simply using the COCOMO equation.

CHAPTER 7

EXPERIMENTS AND RESULTS

A detailed description of the proposed frameworks together with required training algorithms and data format has been discussed in previous chapters. In this chapter we will discuss experimental details and the algorithms to generate artificial datasets, if original historical datasets is not available; in order to train and test the FLS based effort prediction systems created using the framework. Moreover we also provide the experimental results for investigating the effects of parameters and architectures on FLS based effort prediction framework. The results will be compared and discussed based on root mean square relative error (RMSRE) and prediction accuracy.

7.1. Evaluation of Prediction Accuracy

The root mean square relative error (RMSRE) and prediction at level q i.e., PRED(q) are the quantitative measures that enable us compare the prediction accuracy of our training procedure to the actual values.

Suppose we have a set of n projects, let k be the number of them whose mean magnitude of relative error is less than or equal to q. Then we have:

$$PRED(q) = k / n$$

Conte *et al.* [18] suggests that an acceptable level for mean magnitude of relative error is something less than or equal to 0.25. This notion was used to define a measure of prediction quality. For example, if PRED(0.25) = 0.53, then 53% of the predicted values fall within 25% of the original values. It is always desirable to maximize this value and minimize RMSRE [63].

7.2. Experiment 1: Uncertainty Handling in FLS Based Effort Prediction Framework when Size is Provided as a Singleton Input

This experiment deals with the performance evaluation of the proposed framework, discussed in Chapter 5, for handling imprecision and uncertainty in software development effort prediction when size is provided as a singleton input. In the following subsections, we will discuss algorithm for artificial dataset generation, training and testing of the prediction system, and the results.

7.2.1. Algorithm for Artificial Dataset Generation

Historical dataset plays a vital role in training and testing adaptive FLS based prediction systems generated using the framework. These systems need sufficient historical data to get their parameters trained, which may not always be possible. There is no such widely

accepted huge database available in the public domain or the available databases (e.g., ISBSG database [28]) may not fulfill the requirements for training and testing FLS based software development effort prediction systems in the context of uncertainty handling. Therefore, we have provided algorithms to generate artificial datasets that can be used for this task. The algorithms introduce uncertainty by generating noisy size measures. One thing that must be made clear at this point is that our proposed framework for handling uncertainty is general and can accept any mode and size measure as inputs. But due to wide acceptance of COCOMO effort estimation equations in the software engineering community we have employed them in the artificial dataset generation algorithms. In this experiment and all the experiments discussed later, we assume size within the interval [0, 100] KDSI, unless otherwise stated, and mode assumes values for intermediate COCOMO model as described in Chapter 2. Moreover, our algorithms are inspired by artificial data generation algorithm proposed by Ahmed *et al.* [2]. Our algorithms, as discussed, are designed to produce datasets that represent uncertainty in size measure whereas Ahmed *et al.* algorithm does not address this.

In Algorithm 7-1, for each data point we compute noisy size values by first generating a random size value, using uniform distribution, within predefined interval and then we compute uniform random noise that remains within R percent of the generated size value. The generated noise is then added to the size value. In the following experiments we assume data sets with 15% and 25% noise in the size to see how increase in noise and consequently uncertainty affects the effort prediction performance.

Algorithm 7-1: Generate artificial datasets that represent uncertainty in size

1. Generate K unique random size values in the interval $[s^-, s^+]$ using uniform distribution.
2. For each size generated in (1) select any of the three mode values randomly and compute the nominal effort as:

$$\text{Nominal Effort} = A \times [Size]^B$$

3. For each size value generated in (1) generate noise, using uniform distribution, up to at most R percent and add that noise to the corresponding size value. Compute the nominal effort for all the newly generated size values using the above equation and the same mode as was chosen for the corresponding original size value.
4. Repeat step (3) to generate as many noisy {size, nominal effort} pairs for chosen mode values as needed.
5. Take the average of nominal effort values in each data point to have the single nominal effort for all the size values.
6. Partition the K data points into training and testing datasets. The training dataset consists of 75 percent of the entire dataset whereas the remaining dataset is left for testing.

7.2.2. Training and Testing

We have conducted ten experiments using ten different datasets. The first five datasets assume at most 15 percent uncertainty in the size whereas the last five experiments assume at most 25 percent uncertainty in the size. Each dataset consists of 250 data points where each data point contains one mode and four size values associated with a single nominal effort (average of four nominal effort values). In each experiment we have employed type-2 FLS training algorithm as described in Chapter 5 and we have compared the results of this with the corresponding type-1 FLS. It is important to note here that the performance of type-1 and type-2 training algorithms can not be compared

on the same step size[20] because type-1 training algorithm requires a very small value for the step size than type-2. Thus in all five experiments we have provided such step size values to both the FLS types so that parameters are converged on the same number of epochs. Upon the completion of each training epoch, the RMSRE values on both training and testing datasets are computed. After complete training, pred(10) and pred(25) values are also computed to justify the validity of the proposed framework.

7.2.3. Results and Discussion

It is evident from Table 7-1 and 7-2 that none of the type-1 or type-2 fuzzy logic systems has preference over one another on the basis of pred(10) or pred(25). In some experiments type-1 has shown better performance whereas in the other experiments type-2 has got some edge over type-1 FLS. But if we look at the RMSRE graphs we can say that type-2 FLS has outclassed the type-1 FLS. The reason for this is that pred(q) accepts all the outputs that are within *k* percent of error, no matter if some outputs are marginally inside this limit. The RMSRE, on the other hand, describes how close the predicted outputs to the actual outputs are. Since type-2 FLS handles uncertainty better than type-1 FLS therefore the outputs produced by type-2 FLS are much closer to the actual outputs. Therefore our intuition for handling uncertainty in effort prediction frameworks using type-2 FLS is justified. Moreover it can be observed from Table 7-1 and 7-2 that prediction accuracy decreases with the increase in uncertainty.

---

[20] It is the parameter whose value is multiplied with the MF's parameters updates to smoothly tune the prediction system.

Table 7-1: Summary of Prediction Quality using type-1 and type-2 Fuzzy Logic Systems on five different datasets when there is 15 percent uncertainty in Size, showing PRED(10) and PRED(25) on training and testing datasets.

Experiment No.	Type-1 Fuzzy Logic System				Type-2 Fuzzy Logic System			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	0.9600	0.8900	0.8650	0.7950	0.9850	0.9800	0.8750	0.9450
2	0.9800	0.9800	0.8950	0.9200	0.9750	0.9900	0.9000	0.9350
3	0.9750	0.9150	0.8600	0.8650	0.9600	0.9200	0.8100	0.7800
4	0.9700	1.0	0.8900	0.9550	0.9750	1.0	0.8800	0.9400
5	0.9750	1.0	0.8750	0.9150	0.9900	1.0	0.9450	0.9850

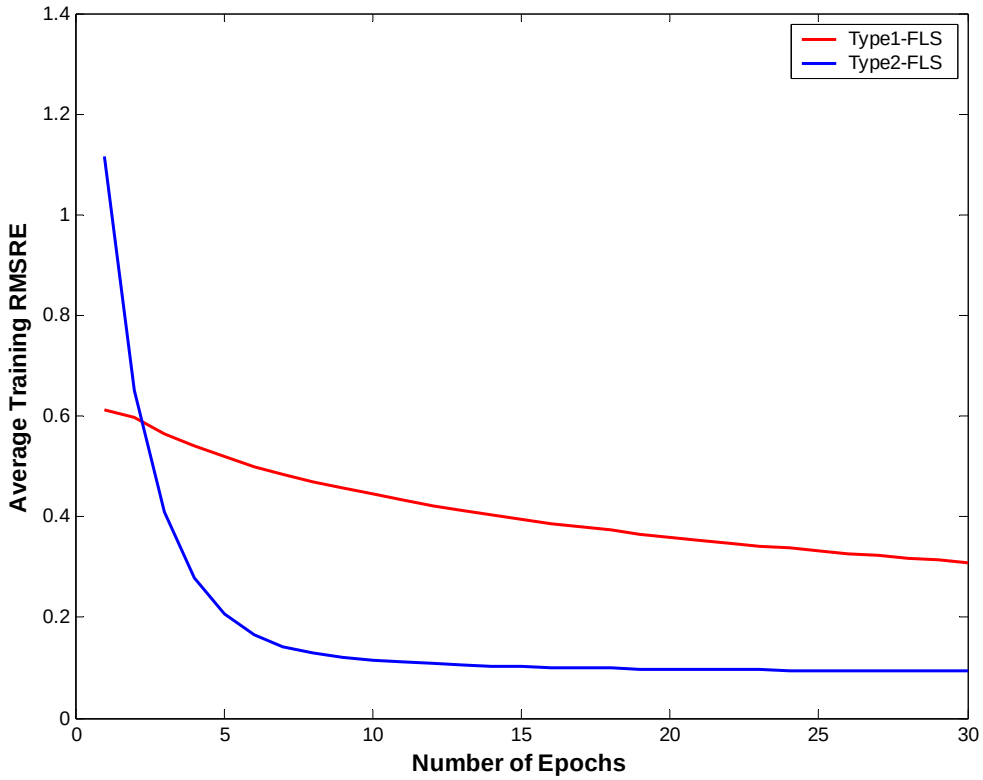

Figure 7-1: Average RMSRE graph of training type-1 and type-2 FLSs on singleton size inputs (15% uncertainty).

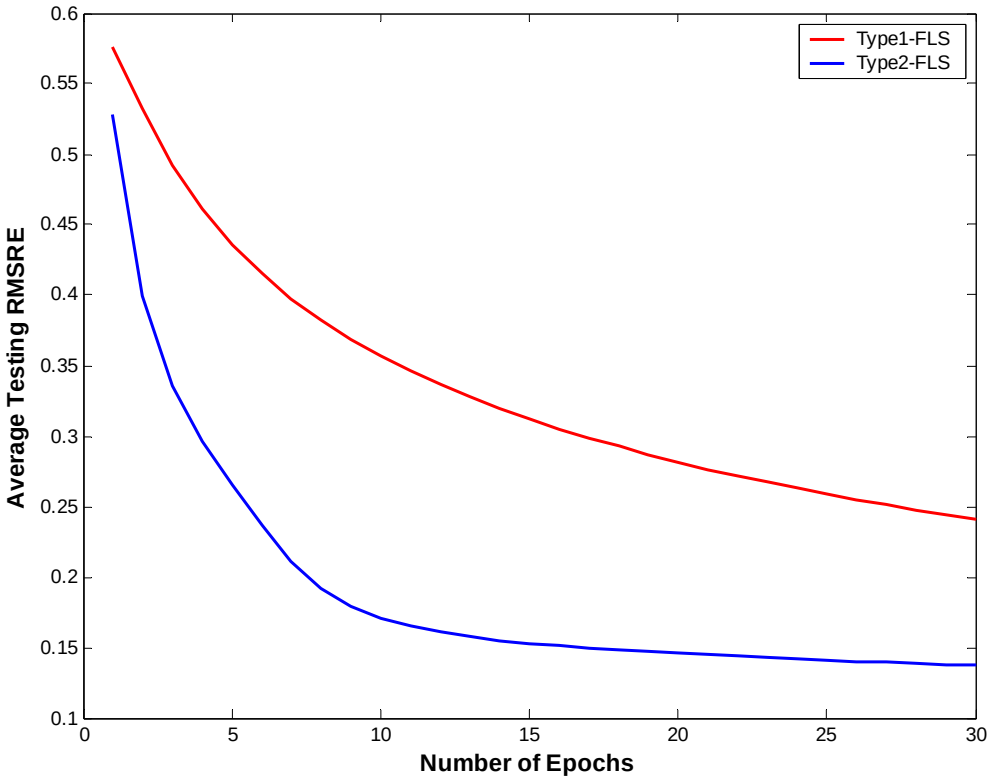

Figure 7-2: Average RMSRE graph of testing type-1 and type-2 FLSs on singleton size inputs (15% uncertainty).

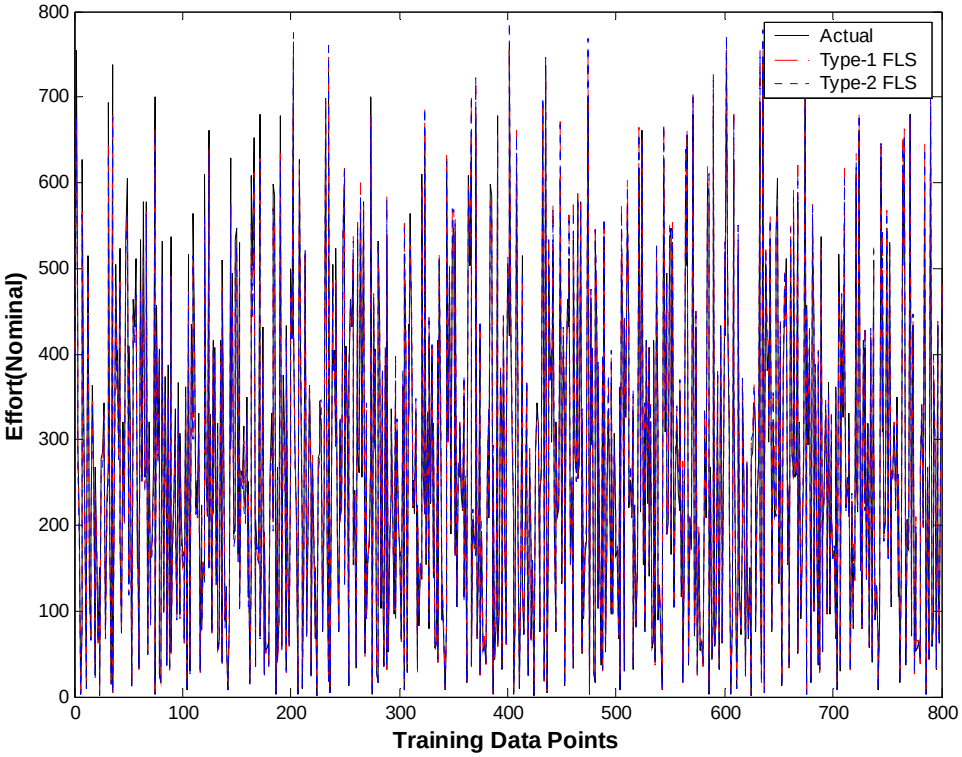

Figure 7-3: Prediction of nominal effort using trained type-1 and type-2 FLSs on training dataset with 15% uncertainty.

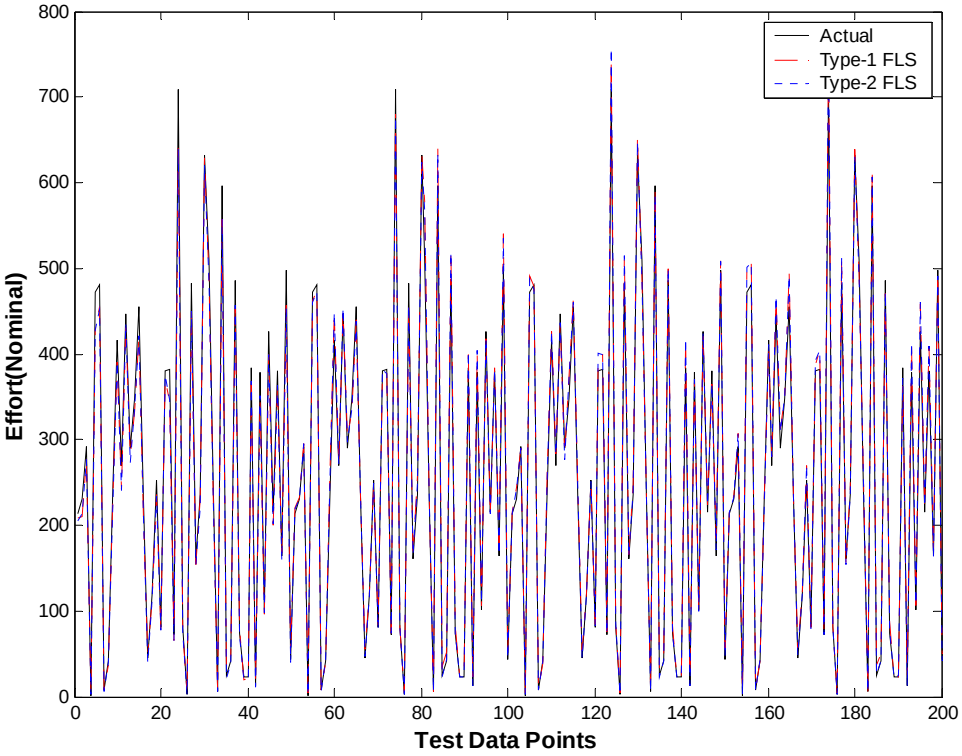

Figure 7-4: Prediction of nominal effort using trained type-1 and type-2 FLSs on test dataset with 15% uncertainty.

Table 7-2: Summary of Prediction Quality using type-1 and type-2 Fuzzy Logic Systems on five different datasets when there is 25 percent uncertainty in Size, showing PRED(10) and PRED(25) on training and testing datasets.

Experiment No.	Type-1 Fuzzy Logic System				Type-2 Fuzzy Logic System			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	0.9800	0.9600	0.4600	0.6500	0.9650	0.9200	0.4300	0.6100
2	0.9750	0.9600	0.3450	0.6700	0.9700	0.9100	0.3950	0.6350
3	0.9650	0.9950	0.4150	0.6800	0.9900	1.0	0.4250	0.7000
4	0.9750	0.9950	0.3750	0.7200	0.9850	0.9950	0.4400	0.7350
5	0.9750	1.0	0.4800	0.7100	0.9850	1.0	0.4350	0.7300

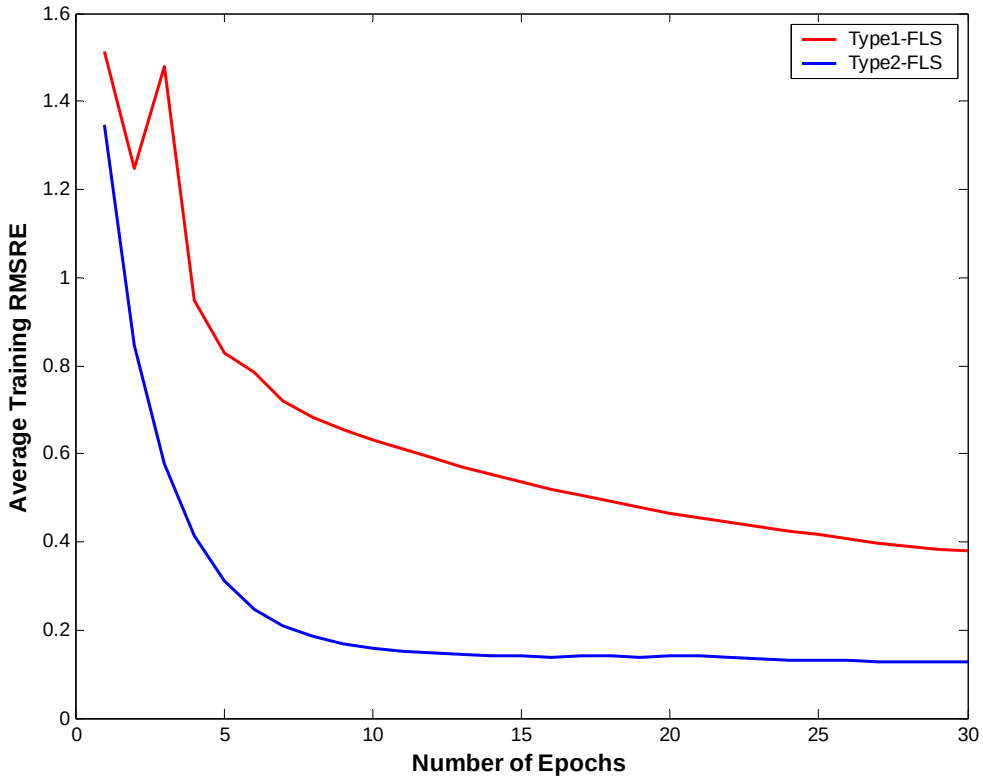

Figure 7-5: Average RMSRE graph of training type-1 and type-2 FLSs on singleton size inputs (25% uncertainty).

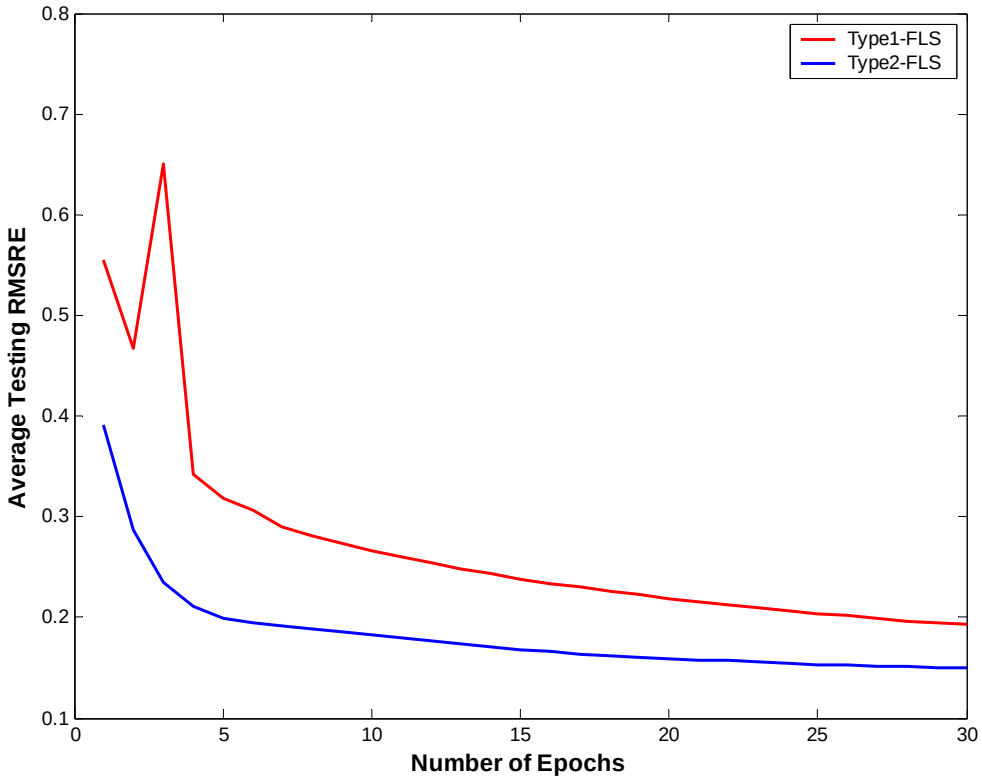

Figure 7-6: Average RMSRE graph of testing type-1 and type-2 FLSs on singleton size inputs (25% uncertainty).

Abrupt changes in the beginning of both training and testing RMSRE curves for type-1 FLS is observed; see Figure 7-5 and 7-6. This is because of applying large values to the step size, which prevents training from being trapped into the local minima.

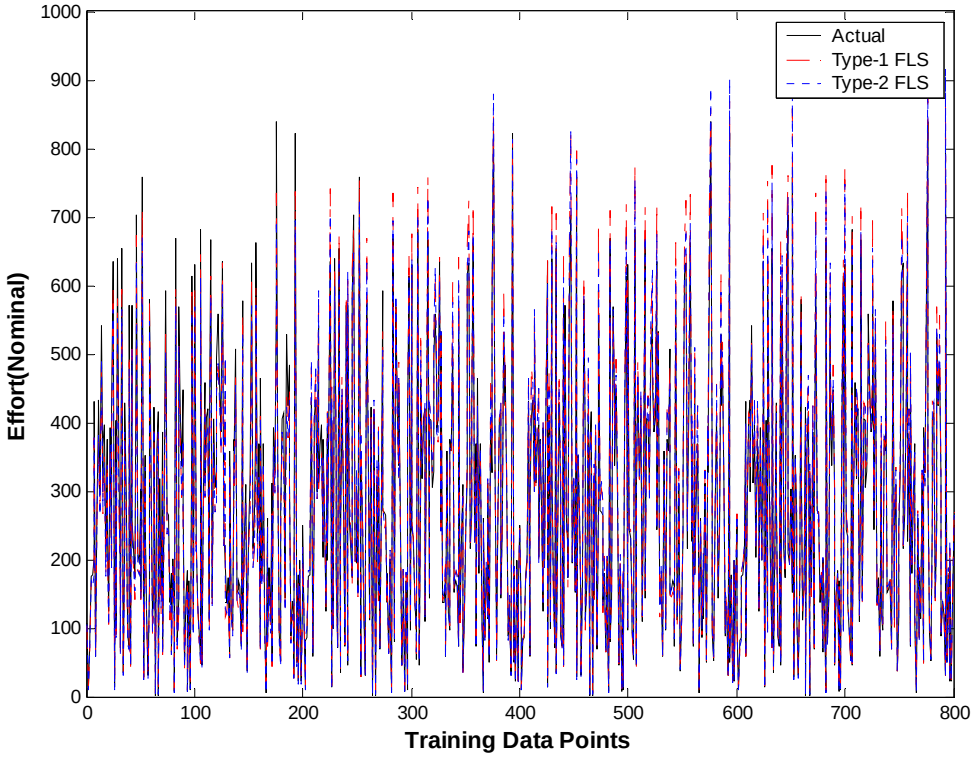

Figure 7-7: Prediction of nominal effort using trained type-1 and type-2 FLSs on training dataset with 25% uncertainty.

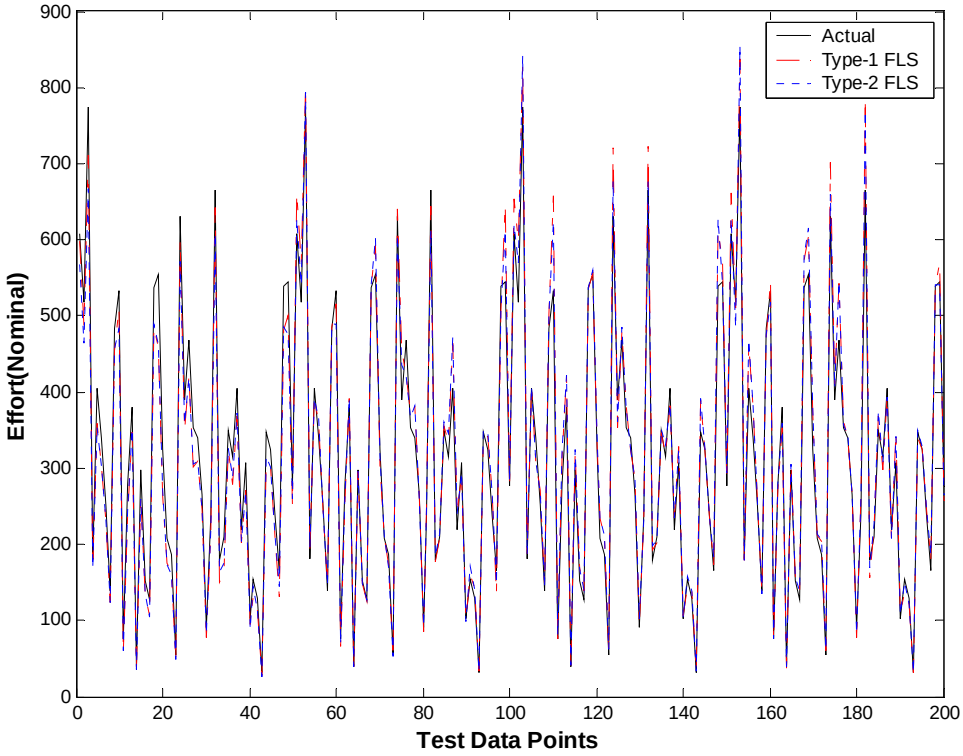

Figure 7-8: Prediction of nominal effort using trained type-1 and type-2 FLSs on test dataset with 25% uncertainty.

7.3. Experiment 2: Uncertainty Handling in FLS Based Effort Prediction Framework when Size is Provided as a Non-Singleton Input

This experiment is concerned with the performance evaluation of the proposed framework, discussed in Chapter 5, for handling imprecision and uncertainty in software development effort prediction when size is provided as a non-singleton input. In the following subsections, we will discuss algorithm for artificial dataset generation, training and testing of the prediction system, and the results.

7.3.1. Algorithm for Artificial Dataset Generation

The algorithm for generating artificial datasets for this system is almost the same as Algorithm 7-1, described earlier, with a little difference that here instead of generating singleton values for size we need to generate Gaussian membership functions that represent size as non-singleton input.

Algorithm 7-2: Generate artificial datasets that represent uncertainty when size is defined using Gaussian membership function.

1. Generate K unique random size values in the interval $[s^-, s^+]$ using uniform distribution. These will serve as the means for the input size membership functions.
2. For each mean size, compute random standard deviation so that Gaussian membership function spans R to Q percent of the mean value on both sides of the mean.
3. For each mean size generated in (1) select any of the three mode values randomly and compute the nominal effort as:

$$\text{Nominal Effort} = A \times [Size]^B$$

4. For each mean size generated in (1) generate uniform noise up to at most R percent and add that noise to the corresponding size value. Compute standard deviations as in (2) and the nominal effort values for all the newly generated size values using the above equation and the same mode as was chosen for the corresponding original size value.
5. Repeat step (4) to generate as many noisy {size, nominal effort} pairs for chosen mode values as needed.
6. Take the average of nominal effort values in each data point to have the single nominal effort for all the size values.
7. Partition the K data points into training and testing datasets. The training dataset consists of 75 percent of the entire dataset whereas the remaining dataset is left for testing.

7.3.2. Training and Testing

We have conducted ten experiments using ten different datasets. The first five datasets assume at most 15 percent uncertainty in the size whereas the last five experiments assume at most 25 percent uncertainty in the size. In all ten experiments, we have generated Gaussian size input membership functions such that they have spanned 25 to 27 percent of the mean value on both sides of the mean. The rest of the detail is the same as we have discussed in experiment 1.

7.3.3. Results and Discussion

It is again evident from Table 7-3 and 7-4 that none of the type-1 or type-2 fuzzy logic systems has preference over one another on the basis of pred(10) or pred(25). Because in some experiments type-1 has shown better performance whereas in the other experiments type-2 has got some edge over type-1 FLS based effort prediction framework. But if we look at the RMSRE graphs we can say that type-2 FLS has outclassed the type-1 FLS. The reason for this behavior is the same as we have discussed in Section 7.2.3 for singleton size inputs. Therefore our intuition for handling uncertainty in effort prediction frameworks using type-2 FLS is justified. Moreover it can be observed from Table 7-3 and 7-4 that prediction accuracy decreases with the increase in uncertainty.

Table 7-3: Summary of Prediction Quality using type-1 and type-2 Fuzzy Logic Systems on five different datasets when there is 15 percent uncertainty in Size where it is defined as a Non-Singleton input, showing PRED(10) and PRED(25) on training and testing datasets.

Experiment No.	Type-1 Fuzzy Logic System				Type-2 Fuzzy Logic System			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	0.9650	0.9400	0.9425	0.9000	0.9825	0.9500	0.9313	0.8950
2	0.9750	0.9600	0.9413	0.9250	0.9850	0.9900	0.9187	0.9000
3	0.9850	0.9950	0.9463	0.9350	0.9825	0.9950	0.9200	0.9500
4	0.9850	1.0	0.9387	0.9750	0.9812	1.0	0.9300	0.9700
5	0.9775	0.9400	0.9163	0.8350	0.9950	0.9800	0.9475	0.9100

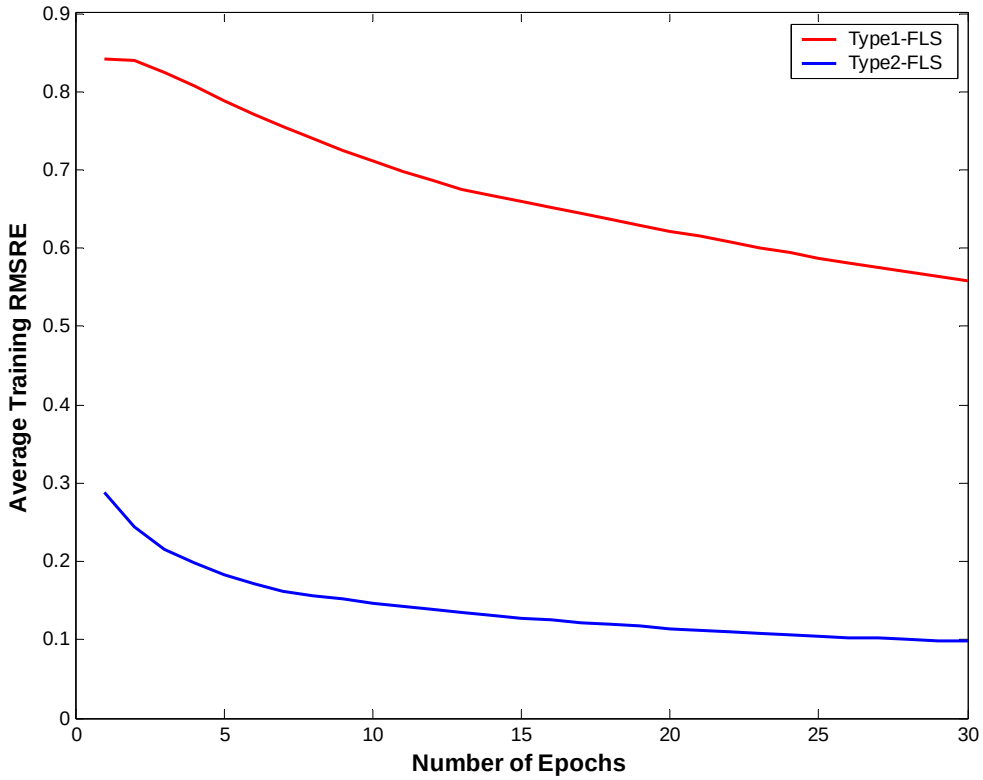

Figure 7-9: Average RMSRE graph of training type-1 and type-2 FLSs on non-singleton size inputs (15% uncertainty).

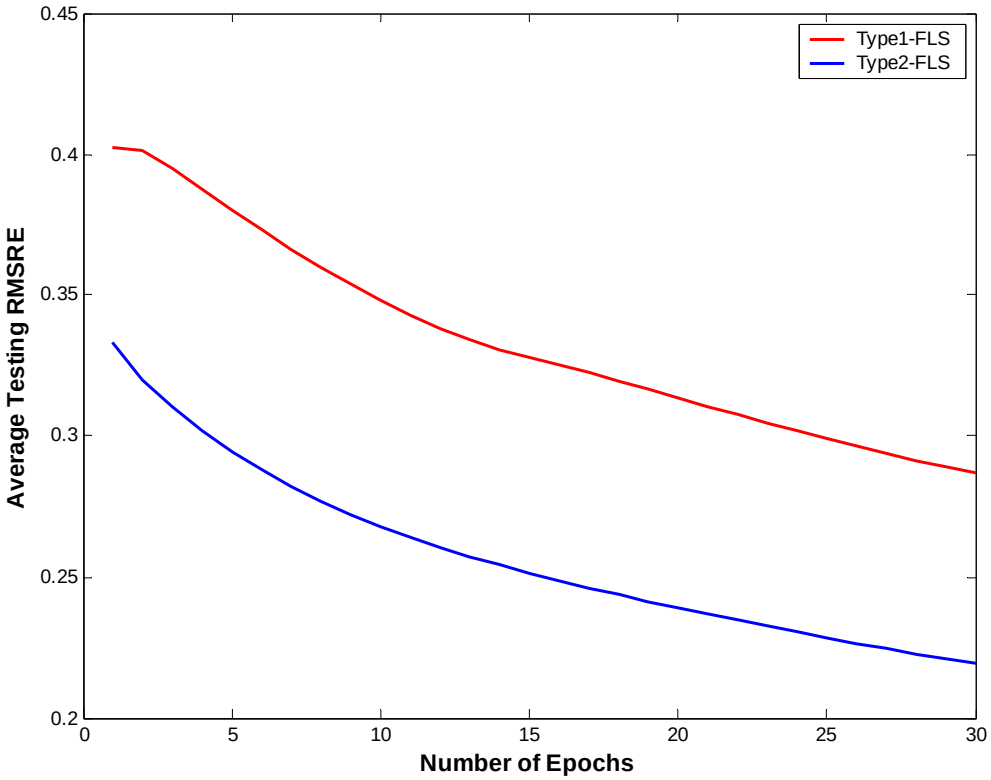

Figure 7-10: Average RMSRE graph of testing type-1 and type-2 FLSs on non-singleton size inputs (15% uncertainty).

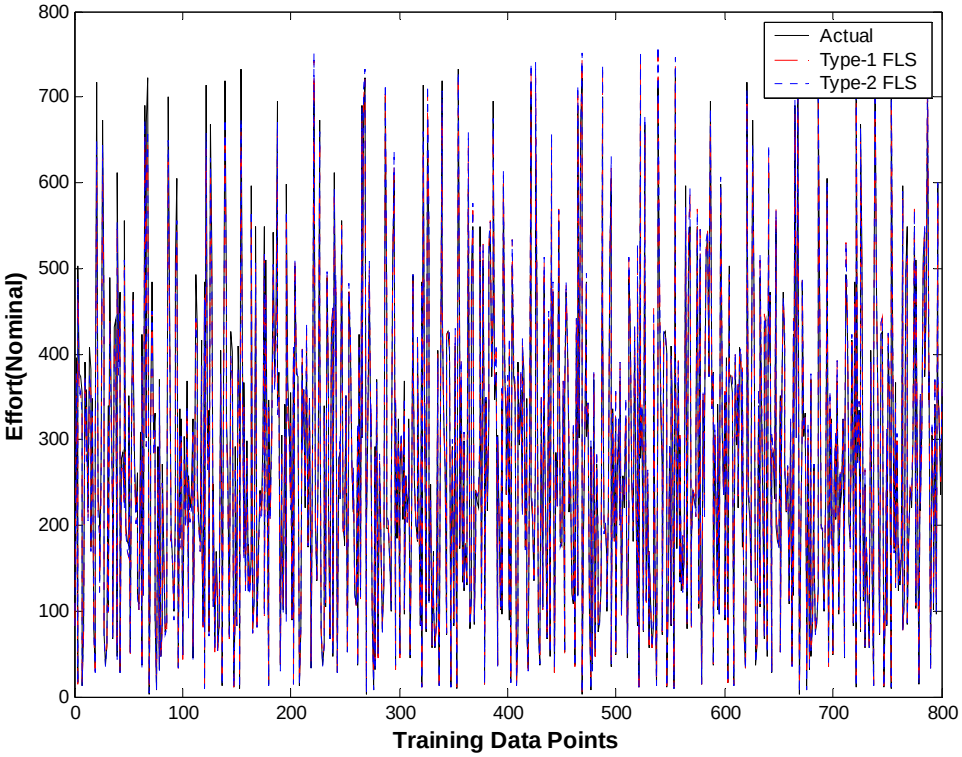

Figure 7-11: Prediction of nominal effort using trained type-1 and type-2 FLSs on training dataset with 15% uncertainty.

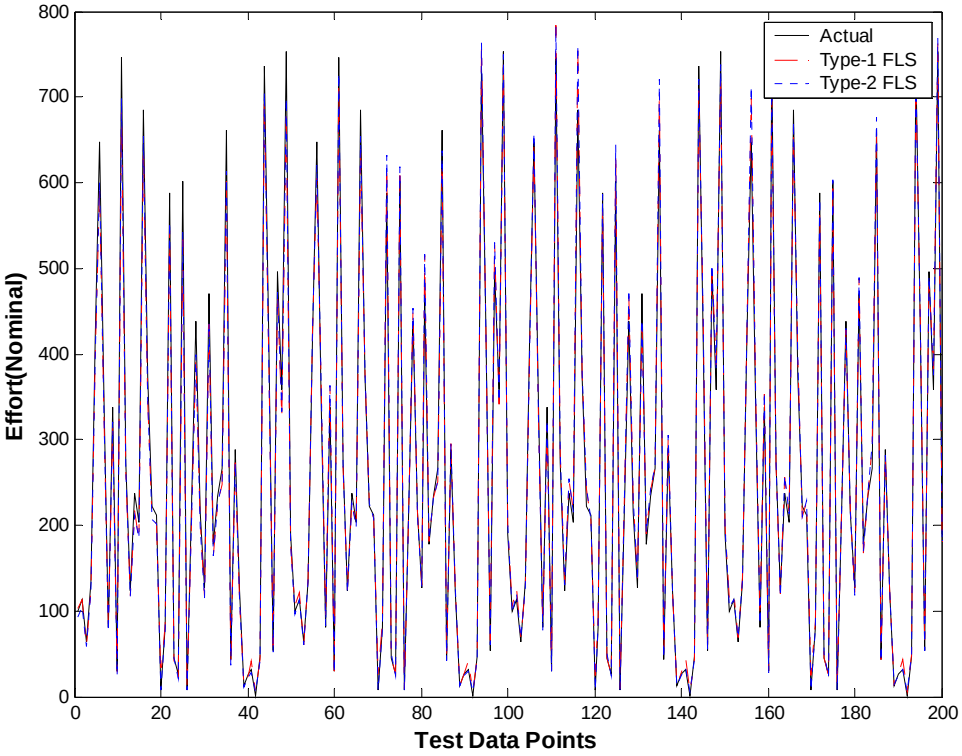

Figure 7-12: Prediction of nominal effort using trained type-1 and type-2 FLSs on test dataset with 15% uncertainty.

Table 7-4: Summary of Prediction Quality using type-1 and type-2 Fuzzy Logic Systems on five different datasets when there is 25 percent uncertainty in Size where it is defined as a Non-Singleton input, showing PRED(10) and PRED(25) on training and testing datasets.

Experiment No.	Type-1 Fuzzy Logic System				Type-2 Fuzzy Logic System			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	0.9600	0.9600	0.6963	0.6950	0.9750	0.9700	0.6775	0.7000
2	0.9463	0.9400	0.6825	0.6500	0.9750	0.9300	0.6775	0.6350
3	0.9812	0.9650	0.7050	0.7150	0.9988	0.9900	0.7050	0.6950
4	0.9675	0.9650	0.6913	0.6700	0.9912	0.9900	0.7050	0.7100
5	0.9425	0.9550	0.6650	0.6200	0.9400	0.9600	0.6700	0.6312

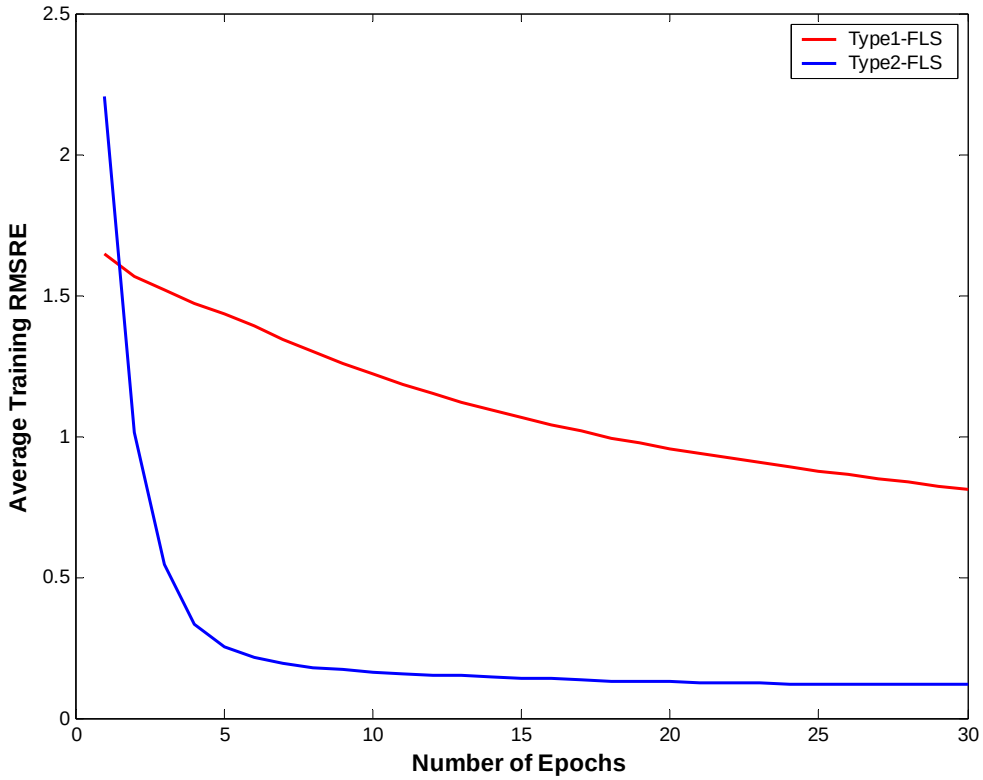

Figure 7-13: Average RMSRE graph of training type-1 and type-2 FLSs on non-singleton size inputs (25% uncertainty).

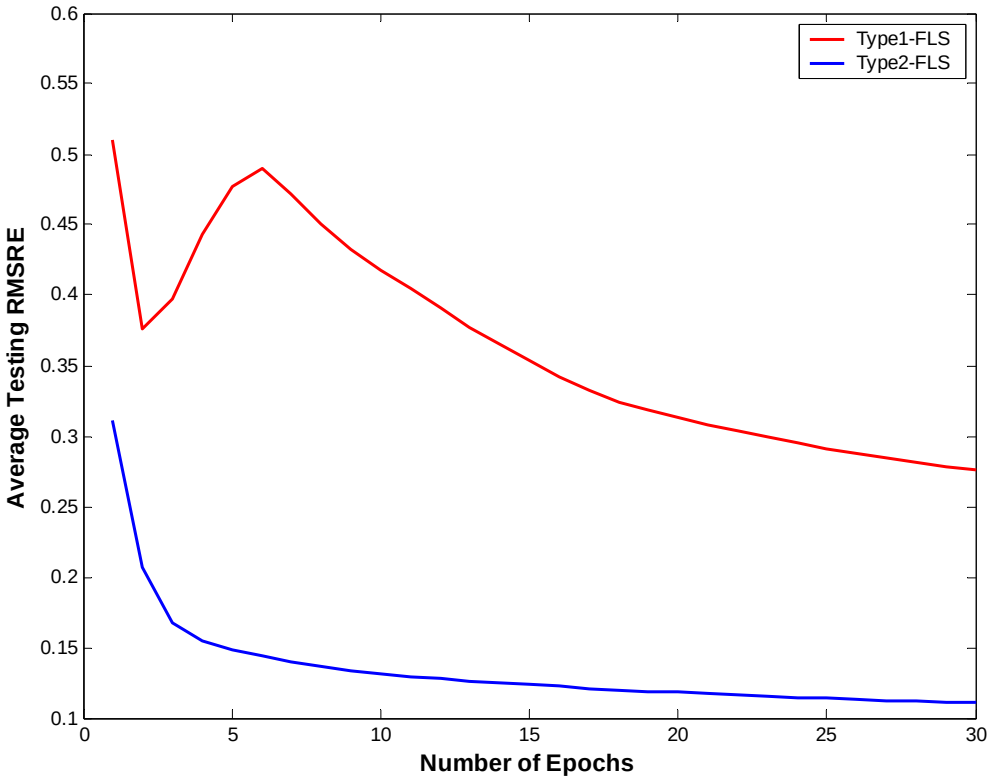

Figure 7-14: Average RMSRE graph of testing type-1 and type-2 FLSs on non-singleton size inputs (25% uncertainty).

An abrupt change in the beginning of the testing RMSRE curve for type-1 FLS is observed; see Figure 7-14. This is because of applying large values to the step size, which prevents training from being trapped into the local minima.

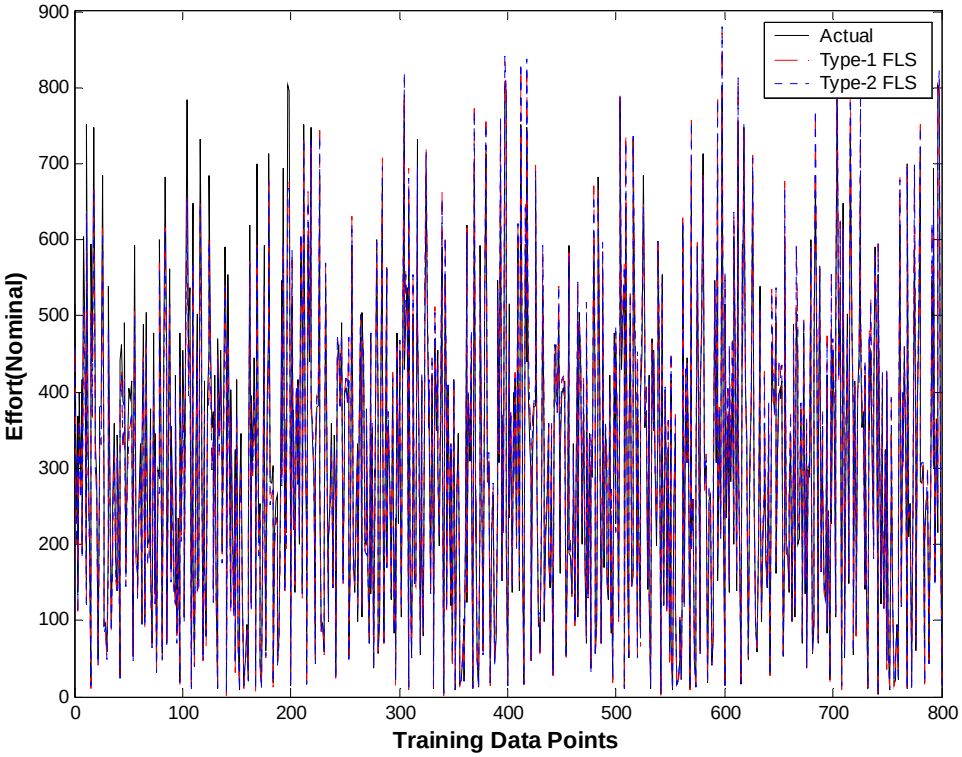

Figure 7-15: Prediction of nominal effort using trained type-1 and type-2 FLSs on training dataset with 15% uncertainty.

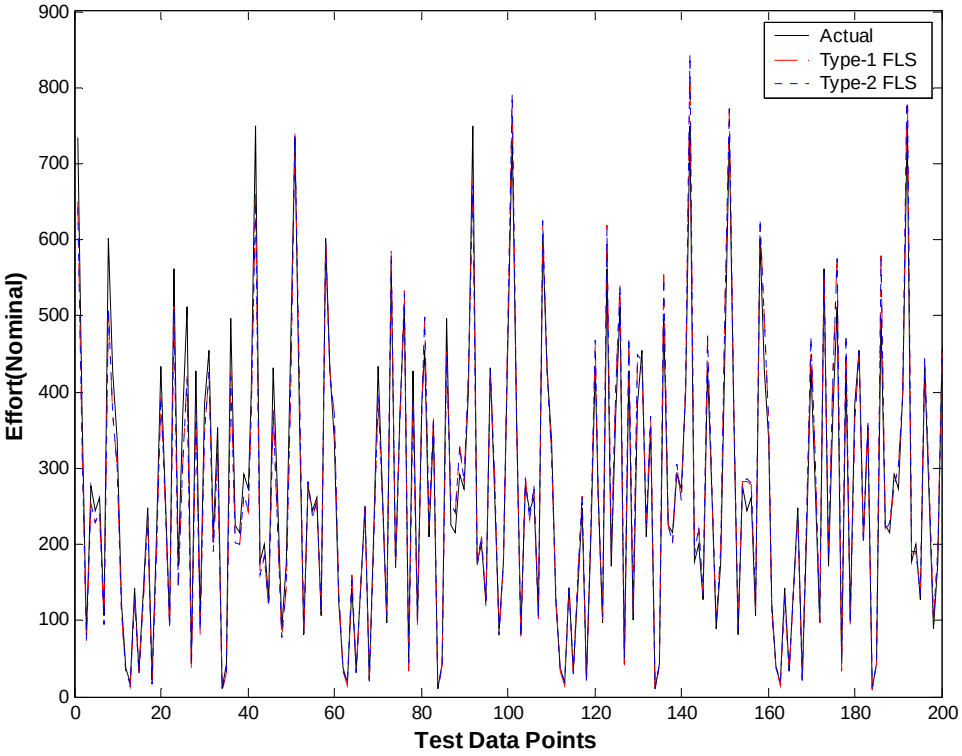

Figure 7-16: Prediction of nominal effort using trained type-1 and type-2 FLSs on test dataset with 15% uncertainty.

7.4. Experiment 3: Handling Uncertainty Due to Laziness/Ignorance in FLS Based Effort Prediction Framework

This experiment deals with the performance evaluation of the proposed framework, discussed in Chapter 5, for handling uncertainty due to laziness/ignorance in software development effort prediction. In the following subsections, we will discuss how one can generate artificial datasets for training and testing of the prediction system. We will also comment on the obtained results.

7.4.1. Algorithm for Artificial Dataset Generation

The logic for generating artificial dataset for arbitrary projects in this experiment is very simple. We define *N* number of projects and generate cost driver values for all the seventeen cost drivers in the [0 1] interval. Then we keep six cost drivers (RELY, CPLX, TIME, ACAP, PCAP and PCON) the same for all the *N* projects and change rest of the cost driver values to produce arbitrary dataset for further N projects. This way we end up with input dataset for desired number of projects. At the end we compute EAF by applying the cost driver values to corresponding FLSs for all the generated projects, see Algorithm 7-3. Consider an example, where N is 50 then executing step (2) of the algorithm will produce dataset for 50 projects. Further, if we execute step (3) five times then we will end up with dataset for 300 projects.

Algorithm 7-3: Generate artificial datasets that represent laziness/ignorance.

1. Define *N* number of projects.
2. Generate cost driver values for all the seventeen cost drivers in [0 1] interval.
3. Keep six cost drivers (RELY, CPLX, TIME, ACAP, PCAP and PCON) values the same for all the *N* projects and generate rest of the cost driver values again.
4. Repeat step (3) until we end up with input dataset for desired number of projects.
5. Compute EAF by applying all the cost driver values to the corresponding FLSs for all the generated projects

7.4.2. Training and Testing

We have conducted five experiments using five different datasets. Each dataset contains 700 data points. According to Algorithm 7-3 the value of N for our datasets is 50. Each dataset is divided into 500 training and 200 testing data points.

7.4.3. Results and Discussion

It is evident from Table 7-5 that type-2 fuzzy logic system is performing better than type-1 when dealing with laziness/ignorance in the input data. The same conclusion can be drawn from RMSRE graphs where we can say that type-2 FLS has outclassed the type-1 FLS. Therefore our intuition for handling laziness/ignorance in effort prediction frameworks using type-2 FLS is justified.

Table 7-5: Summary of Prediction Quality using type-1 and type-2 Fuzzy Logic Systems on five different datasets that represent laziness/ignorance in the cost drivers data, showing PRED(10) and PRED(25) on training and testing datasets.

Experiment No.	Type-1 Fuzzy Logic System				Type-2 Fuzzy Logic System			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	0.5280	0.4800	0.2000	0.2200	0.5220	0.5300	0.2240	0.2550
2	0.4820	0.4300	0.1860	0.1500	0.4900	0.4700	0.1940	0.1800
3	0.4480	0.3650	0.1780	0.1350	0.4820	0.4250	0.1900	0.1600
4	0.4480	0.4650	0.1680	0.1700	0.4920	0.5300	0.1920	0.2100
5	0.4940	0.4100	0.1920	0.1750	0.5340	0.4650	0.2120	0.1957

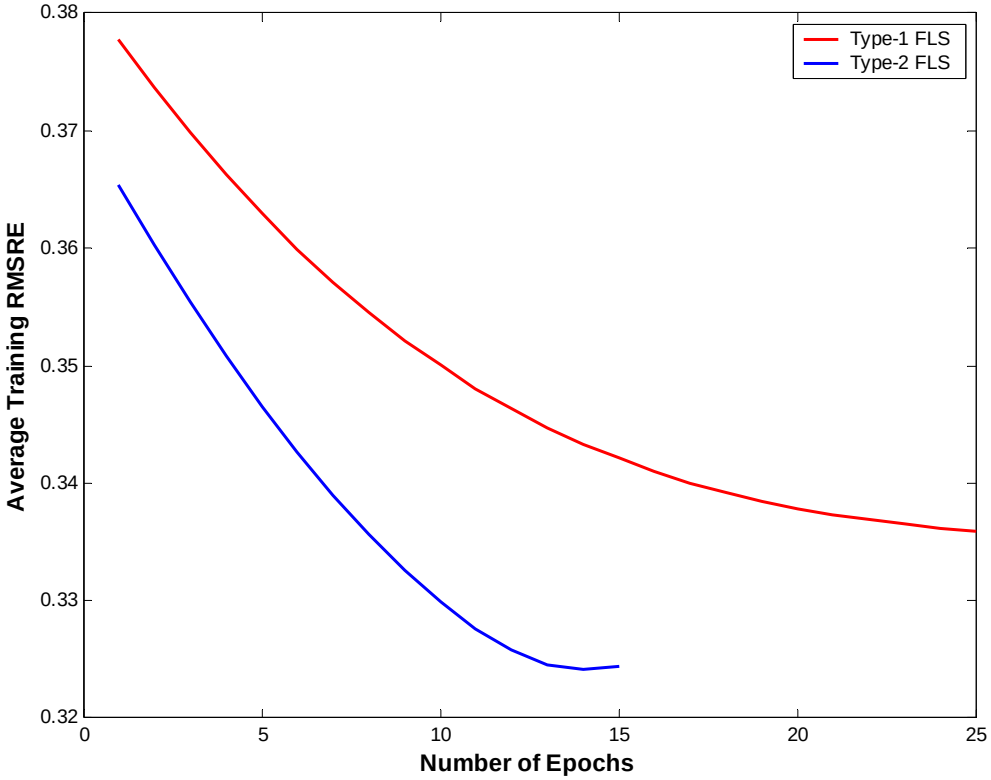

Figure 7-17: Average RMSRE graph of training type-1 and type-2 FLSs for handling laziness/ignorance.

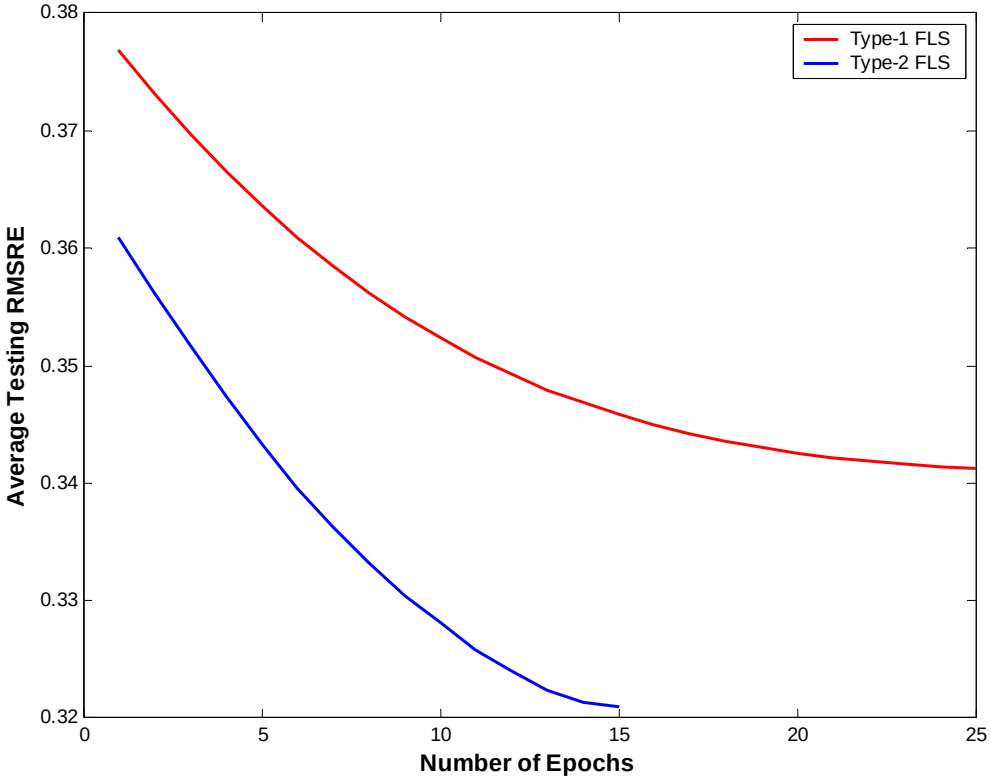

Figure 7-18: Average RMSRE graph of testing type-1 and type-2 FLSs for handling laziness/ignorance.

7.5. Experiment 4: Investigating the Effects of Parameters on Type-1 FLS Based Effort Prediction Framework

This experiment is concerned with the investigation of the effects of various parameters of type-1 FLS on effort prediction. The parameters investigated are: defuzzification method (height vs. modified height), shape of membership function (Gaussian vs. triangular) and propagated error (relative vs. normalized). In the following subsections, we will discuss how one can generate artificial datasets for training and testing of the prediction systems. We will also comment on the obtained results.

7.5.1. Algorithm for Artificial Dataset Generation

In these experiments we have used the algorithm proposed by Saliu [63] for generating artificial datasets, Algorithm 7-4.

Algorithm 7-4: Generate artificial datasets for studying effects of parameters

1. Generate K unique random size values in the interval $[s^-, s^+]$.
2. For each size generated in (1) select any of the three mode values randomly and compute the nominal effort as:
$$\text{Nominal Effort} = A \times [Size]^B$$
3. Partition the K data points into training and testing datasets. The training dataset consists of 70 percent of the entire dataset whereas the remaining dataset is left for testing.

7.5.2. Training and Testing

We have conducted 5 experiments using 5 different datasets for investigation each parameter e.g., defuzzification methods, propagated error and shape of MFs. Each dataset consists of 250 data points, which is divided into 175 training and 75 testing data points. Moreover the experiments are performed in two sets. In the first set of experiments project size lies within [0, 100] KDSI interval while in the other set of experiments it lies within [0, 200] KDSI.

7.5.3. Comparing Height and Modified Height Defuzzification

In this subsection, we provide and discuss the results that are obtained when prediction systems with height and modified height defuzzification methods are trained and tested on artificially generated datasets.

Results and discussion

Analyzing Table 7-6 and 7-7 obtained by computing pred(10) and pred(25) and the corresponding RMSRE graphs, we have come to the conclusion that modified height defuzzification has superiority over height defuzzification method. The reason for this is that unlike height defuzzification, modified height defuzzification method considers the spread of consequent membership function. The spread of consequent portrays a better picture of the contribution of a particular fuzzified input to the corresponding consequent.

Table 7-6: Summary of Prediction Quality using Height and Modified Height Defuzzification methods on five different datasets, showing PRED(10) and PRED(25) on training and testing datasets. Interval of size is [0, 100] KDSI.

Experiment No.	Height Defuzzification				Modified Height Defuzzification			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	1.0	1.0	0.972	0.935	0.9886	0.9771	0.9867	0.9733
2	0.9714	0.9333	0.8571	0.72	0.9943	1.0	0.9829	0.9867
3	0.9486	0.96	0.8171	0.8533	0.9886	1.0	0.9600	0.9867
4	0.9543	0.9467	0.88	0.8267	1.0	1.0	0.9943	0.9867
5	0.9429	0.96	0.8514	0.84	0.9771	1.0	0.9771	1.0

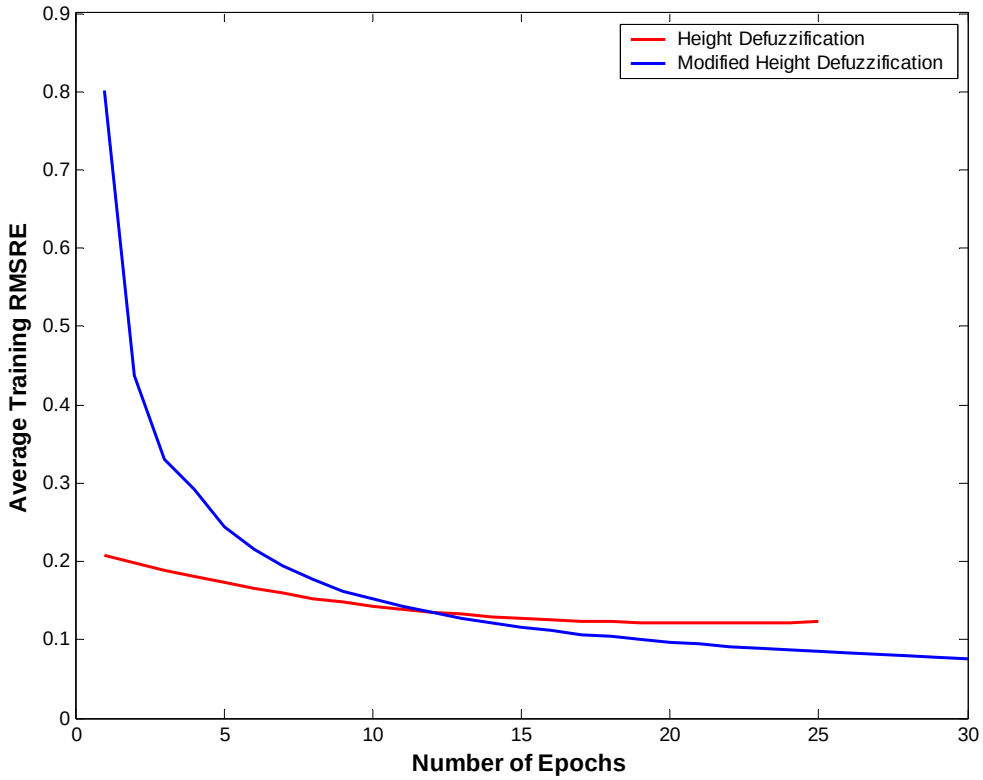

Figure 7-19: Average RMSRE graph of training FLSs with height and modified height defuzzification methods when size interval is [0, 100] KDSI.

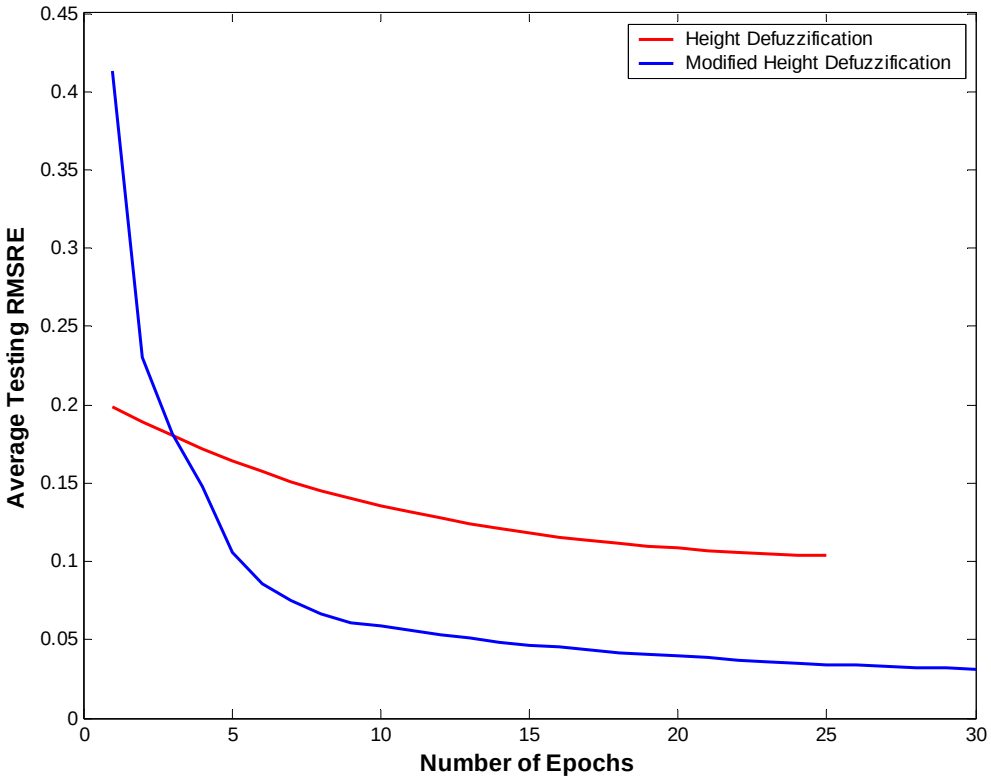

Figure 7-20: Average RMSRE graph of testing FLSs with height and modified height defuzzification methods when size interval is [0, 100] KDSI.

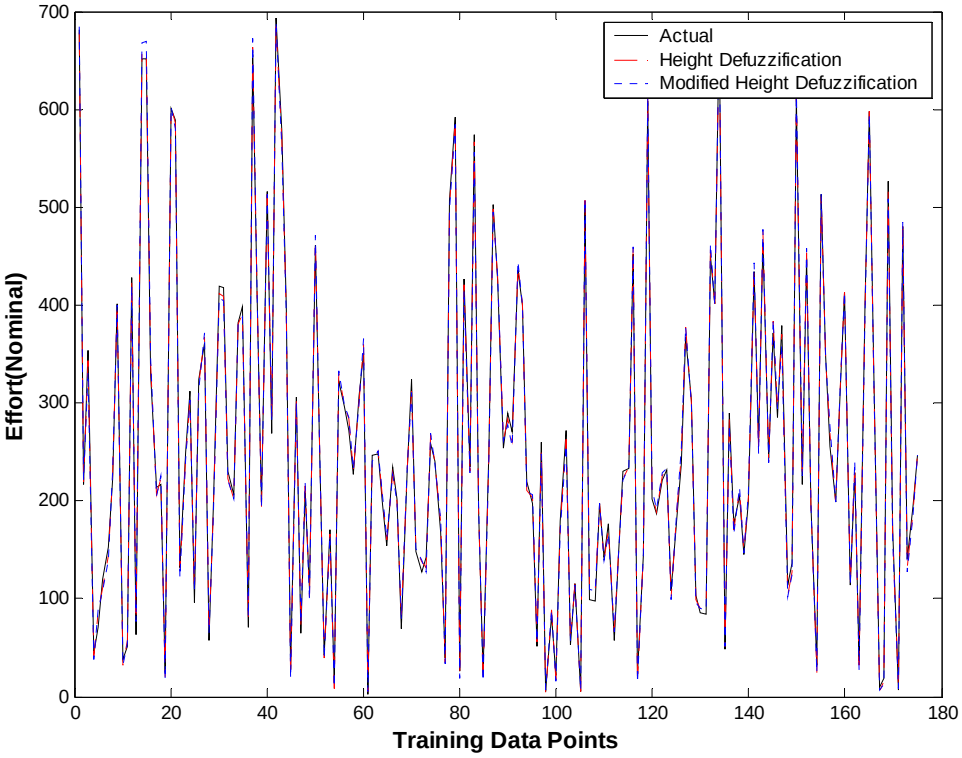

Figure 7-21: Prediction of nominal effort using trained FLSs with height and modified height defuzzification on training dataset.

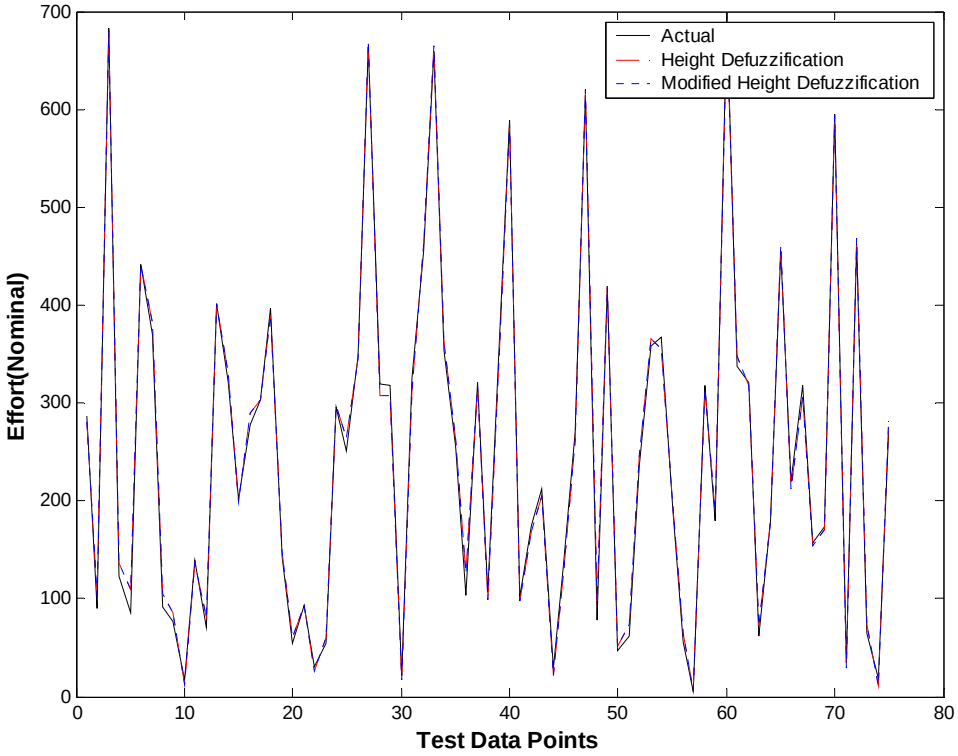

Figure 7-22: Prediction of nominal effort using trained FLSs with height and modified defuzzification on test dataset.

Table 7-7: Summary of Prediction Quality using Height and Modified Height Defuzzification methods on five different datasets, showing PRED(10) and PRED(25) on training and testing datasets. Interval of size is [0, 200] KDSI.

Experiment No.	Height Defuzzification				Modified Height Defuzzification			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	0.9950	1.0	0.9700	0.9500	0.9950	1.0	0.9700	0.9300
2	0.9850	0.9200	0.9600	0.9100	0.9850	0.9700	0.9700	0.9500
3	0.9500	0.8800	0.8400	0.8200	0.9850	0.9800	0.9850	0.9800
4	0.9950	1.0	0.9600	0.9400	0.9950	1.0	0.9850	0.9900
5	0.9950	0.9900	0.9800	0.9500	0.9950	0.9700	0.9800	0.9500

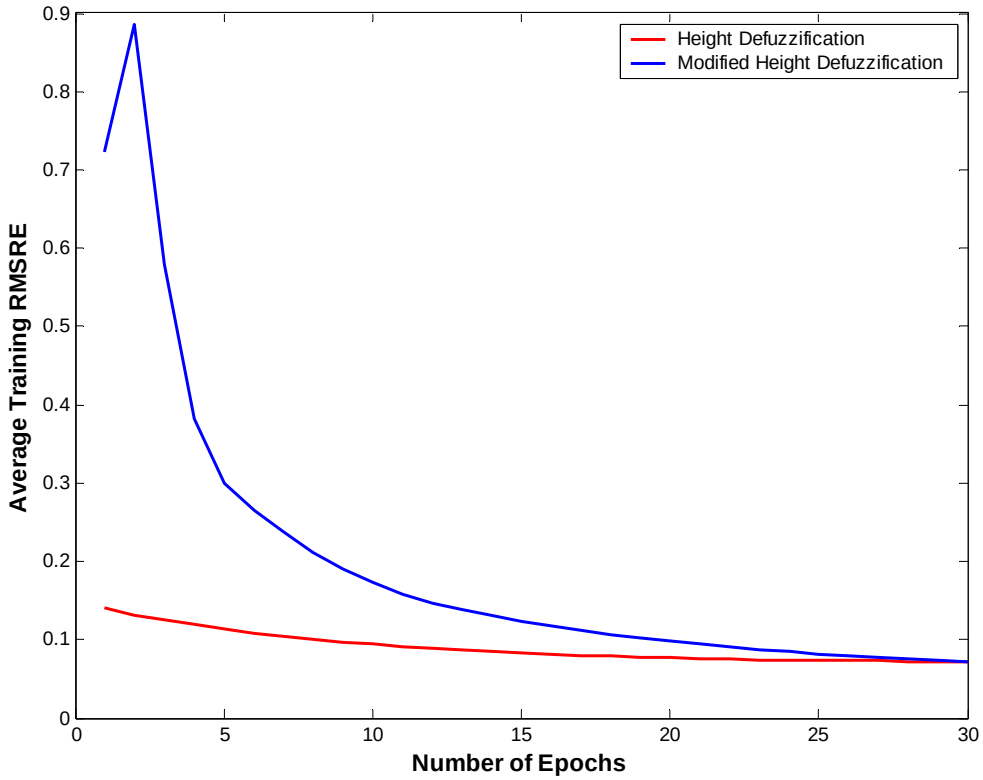

Figure 7-23: Average RMSRE graph of training FLSs with height and modified height defuzzification methods when size interval is [0, 200] KDSI.

The training of FLS with modified height defuzzification requires more iterations for convergence than one with height defuzzification; see Figure 7-23. This is because modified height defuzzification, unlike height defuzzification, has got an additional parameter (spread of consequent MF).

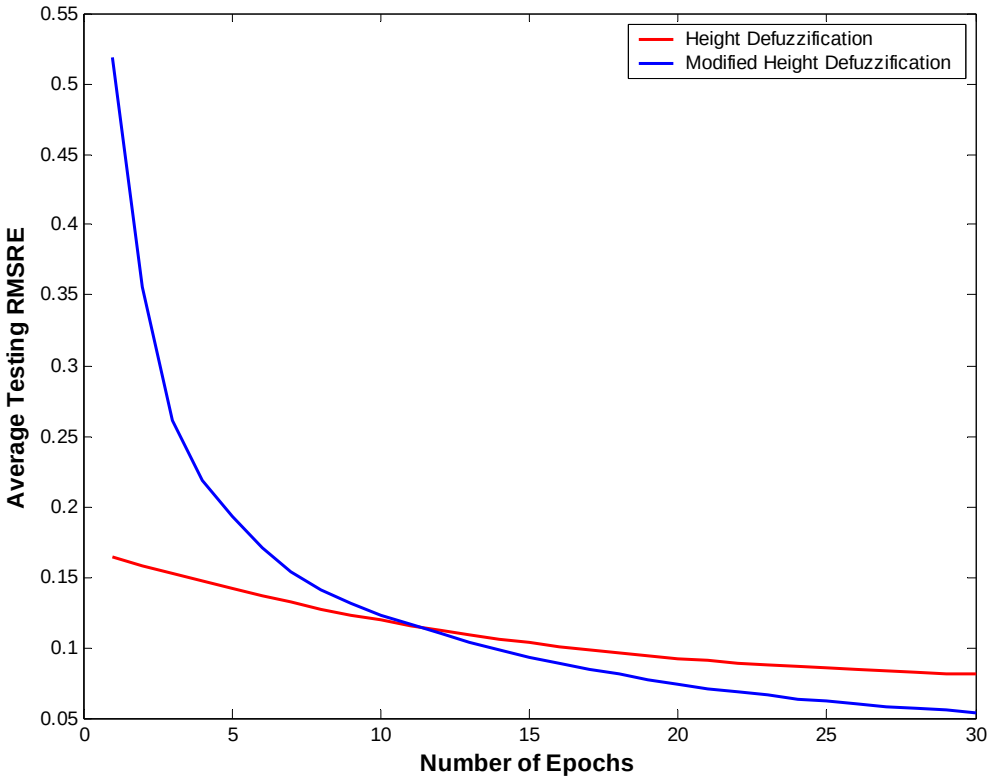

Figure 7-24: Average RMSRE of testing FLSs with height and modified height defuzzification methods when size interval is [0, 200] KDSI.

7.5.4. Comparing Normalized and Relative Error

In this subsection, we provide and discuss the results that are obtained when prediction systems with normalized and relative errors are trained and tested on artificially generated datasets.

Results and Discussion

Analyzing Table 7-8 and 7-9 obtained by computing pred(10) and pred(25) and the corresponding RMSRE graphs, we have come to the conclusion that relative and normalized errors have shown better performance than one another during training and testing, respectively, for size interval [0 100]. But when size interval is increased i.e., [0 200], it is observed that relative error has out performed normalized error during both training and testing. This is due to the fact that using the relative error actually computes the percentage of error for a particular effort value which is supposed to be local to the corresponding data point. Therefore, relative error depicts how much modification to membership functions is needed in order to accommodate the data point. On the other hand, using the normalized error normalizes the effort value using the whole interval of the effort, which can be considered as a global phenomenon.

Table 7-8: Summary of Prediction Quality using Normalized and Relative Error on five different datasets, showing PRED(10) and PRED(25) on training and testing datasets. Interval of size is [0, 100] KDSI.

Experiment No.	Normalized Error				Relative Error			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	0.9771	0.9733	0.8914	0.8267	0.98857	0.97333	0.92	0.82667
2	0.9714	0.9333	0.8571	0.72	0.96571	0.92	0.83429	0.73333
3	0.9486	0.96	0.8171	0.8533	0.91429	0.93333	0.70857	0.81333
4	0.9543	0.9467	0.88	0.8267	0.97143	0.94667	0.80571	0.78667
5	0.9429	0.96	0.8514	0.84	0.94857	0.96	0.81143	0.81333

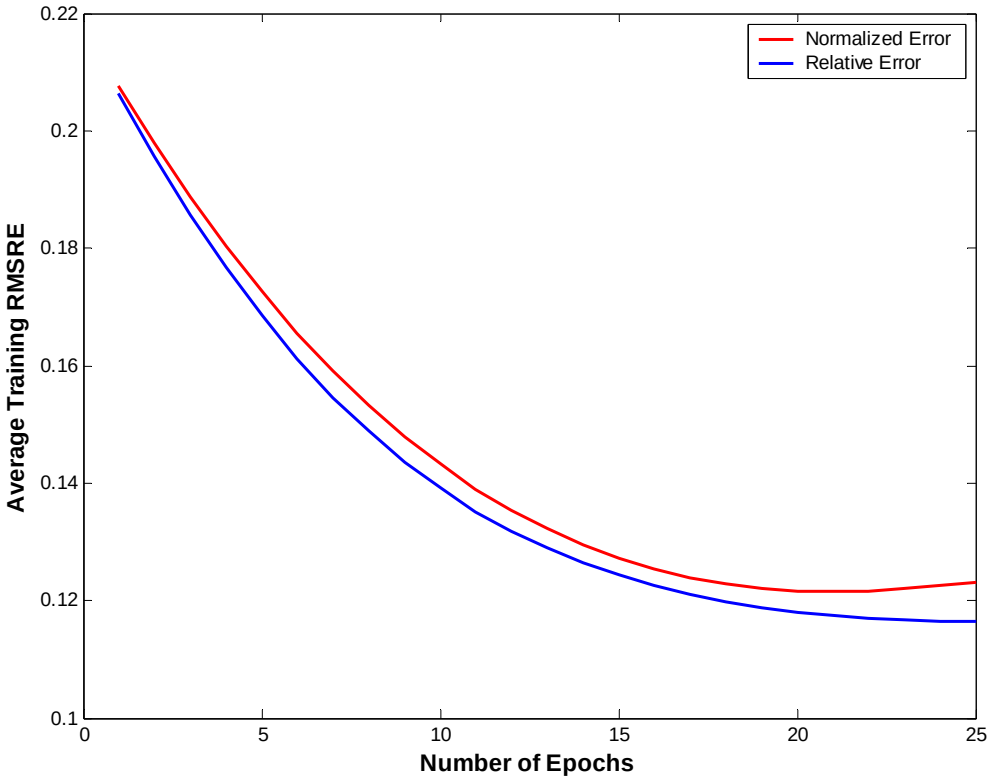

Figure 7-25: Average RMSRE graph of training FLSs with normalized and relative error when size interval is [0, 100] KDSI.

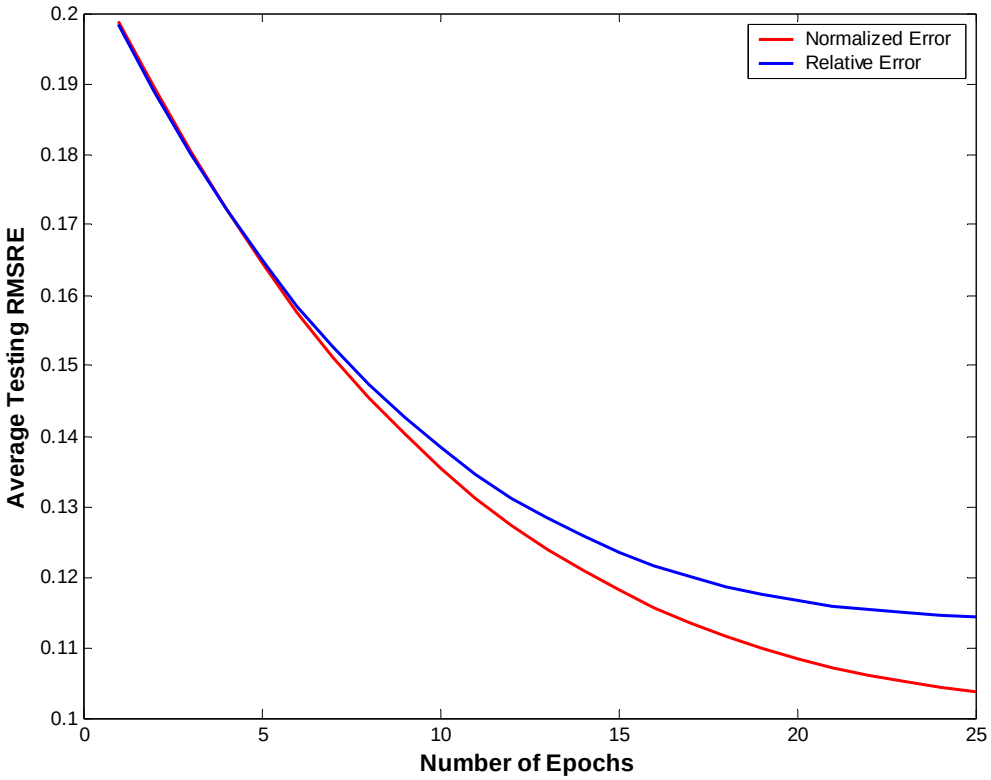

Figure 7-26: Average RMSRE graph of testing FLSs with normalized and relative error when size interval is [0, 100] KDSI.

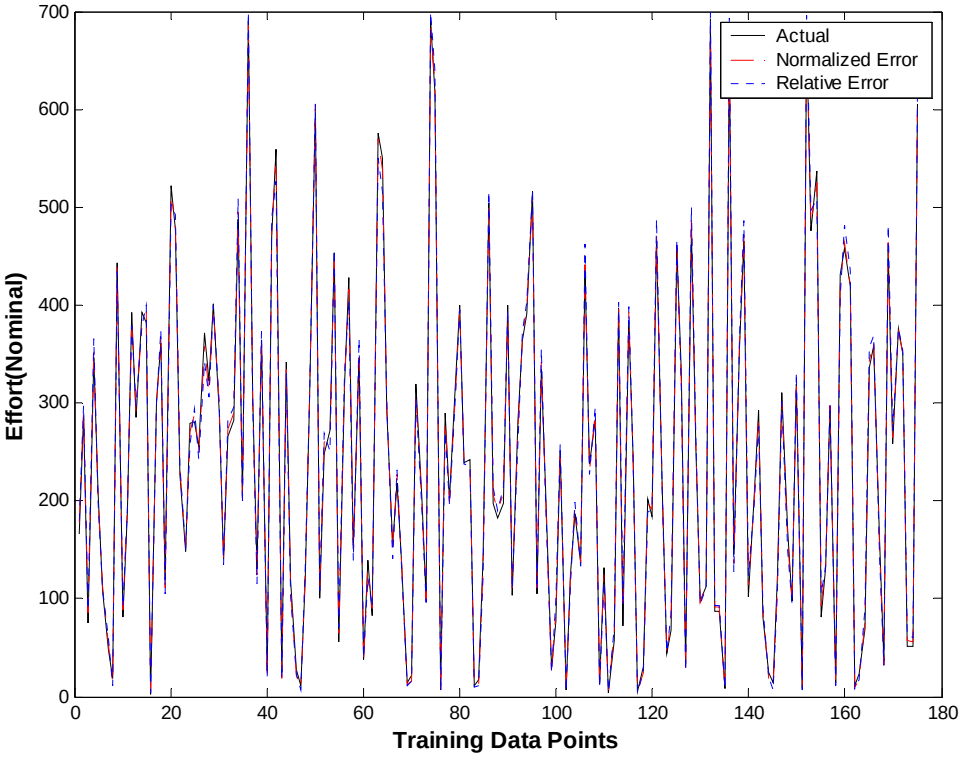

Figure 7-27: Prediction of nominal effort using trained FLSs with normalized and relative error on training dataset.

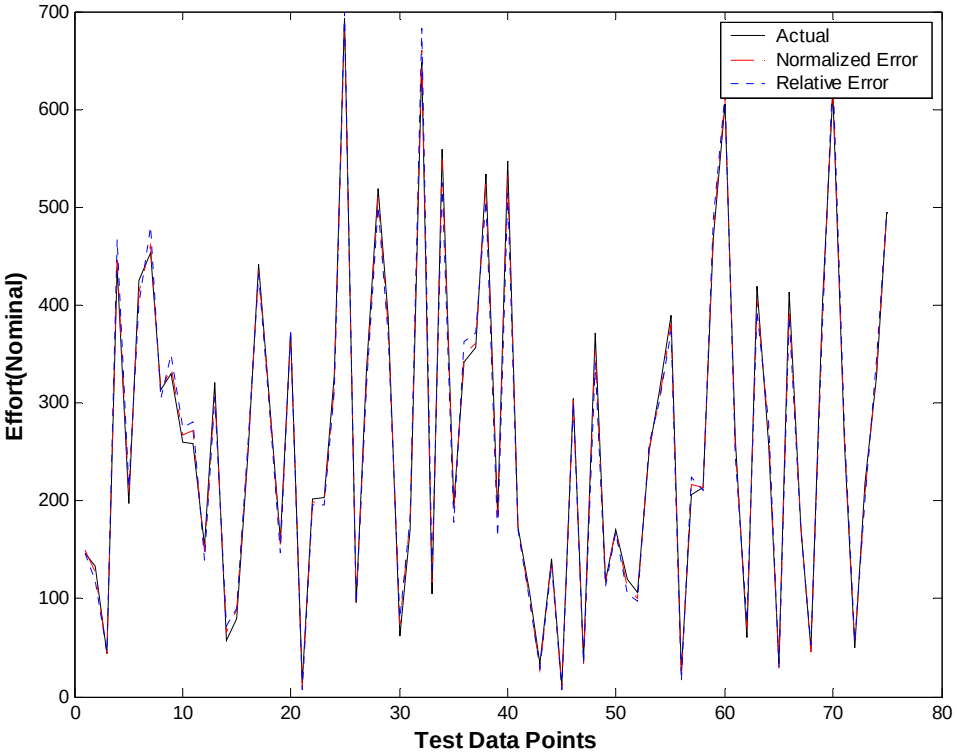

Figure 7-28: Prediction of nominal effort using trained FLSs with normalized and relative error on test dataset.

Table 7-9: Summary of Prediction Quality using Normalized and Relative Error on five different datasets, showing PRED(10) and PRED(25) on training and testing datasets. Size interval is [0, 200] KDSI.

Experiment No.	Normalized Error				Relative Error			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	0.9950	1.0	0.9700	0.9500	0.9950	0.9900	0.9800	0.9700
2	0.9450	0.9200	0.8750	0.8800	0.9900	0.9300	0.9750	0.8700
3	0.9500	0.8800	0.8400	0.8200	0.9600	0.9800	0.9400	0.9200
4	0.9950	1.0	0.9600	0.9400	0.9950	1.0	0.9850	0.9700
5	0.9950	0.9900	0.9800	0.9500	1.0	0.9900	0.9900	0.9600

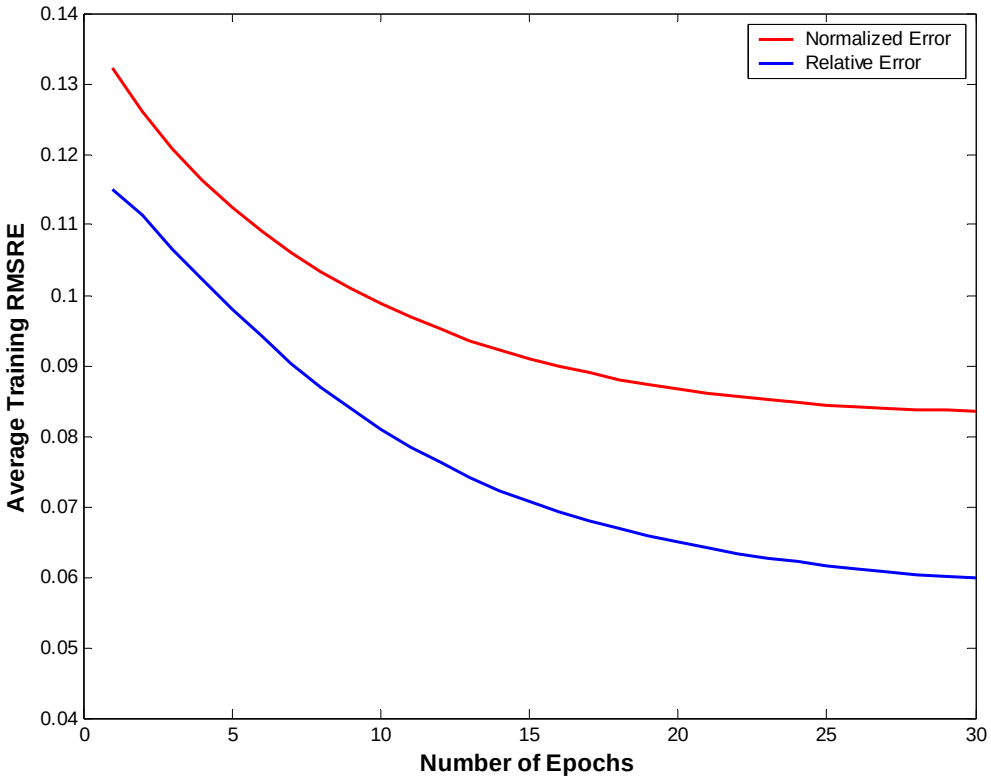

Figure 7-29: Average RMSRE graph of training FLSs with normalized and relative error when size interval is [0, 200] KDSI.

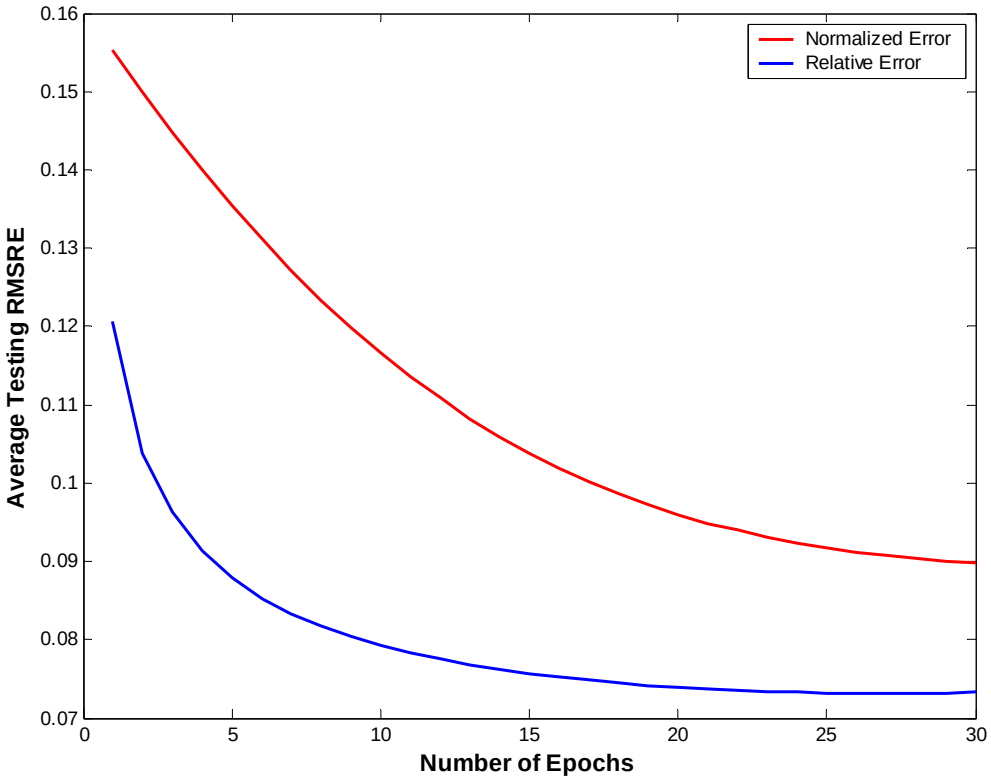

Figure 7-30: Average RMSRE graph of testing FLSs with normalized and relative error when size interval is [0, 200] KDSI.

7.5.5. Comparing Gaussian and Triangular Membership Functions

In this subsection, we provide and discuss the results that are obtained when prediction systems with Gaussian and triangular membership functions are trained and tested on artificially generated datasets.

Results and Discussion

Analyzing Table 7-10 and 7-11 obtained by computing pred(10) and pred(25) and the corresponding RMSRE graphs, we have come to the conclusion that fuzzy logic-based effort prediction system that uses triangular MFs results in better accuracy than one that uses Gaussian MFs. This is due to the locality of changes that the triangular MF provides over the shape (i.e. spread) of the membership function. With Gaussian MFs, regardless of whether an input is fuzzified to the left or to the right of the mean the modification to the spread of the MF is applied equally to both the sides of the mean. In the case of triangular MFs, however, a modification to the spread of the MF is applied only to the side that got fuzzified with the input. Although this modification also affects the other side to some extent, it is still far less than what we have in the case of Gaussian MFs.

Table 7-10: Summary of Prediction Quality using Gaussian and Triangular Membership Functions on five different datasets, showing PRED(10) and PRED(25) on training and testing datasets. Size interval is [0, 100] KDSI.

Experiment No.	Gaussian Membership Function				Triangular Membership Function			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	0.9771	0.9733	0.8914	0.8267	1.0	1.0	0.9486	0.9467
2	0.9714	0.9333	0.8571	0.72	1.0	0.9867	0.9429	0.9067
3	0.9486	0.96	0.8171	0.8533	0.9943	1.0	0.9143	0.9467
4	0.9543	0.9467	0.88	0.8267	1.0	1.0	0.9429	0.88
5	0.9429	0.96	0.8514	0.84	1.0	1.0	0.9543	0.9333

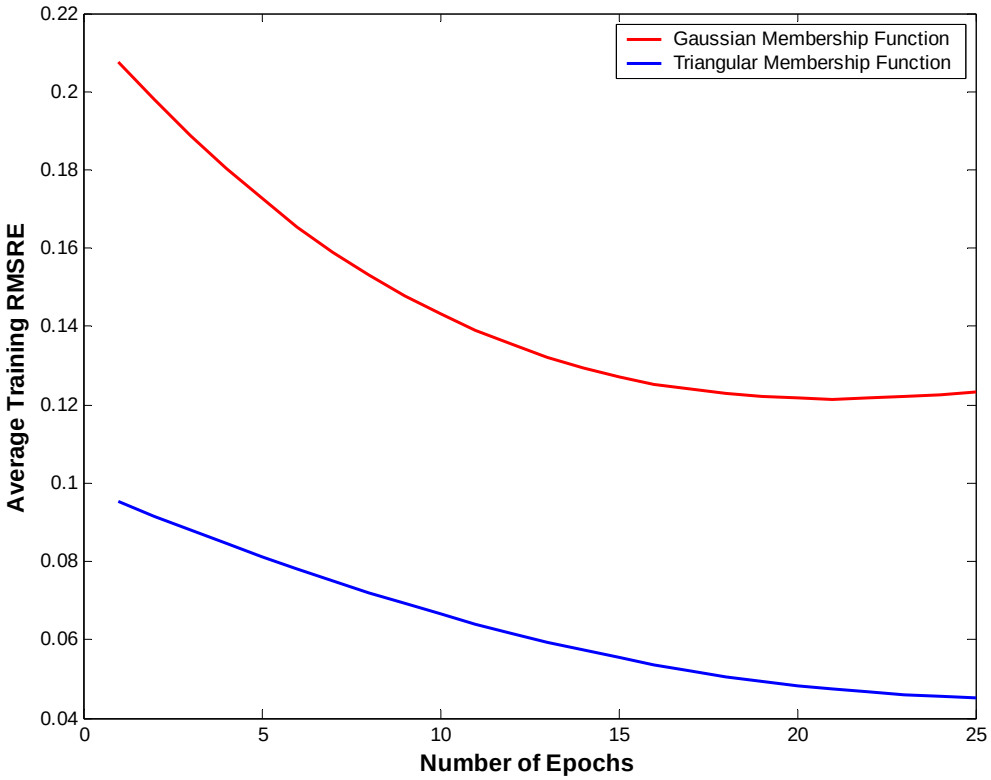

Figure 7-31: Average RMSRE graph of training FLSs with Gaussian and triangular membership functions when size interval is [0, 100] KDSI.

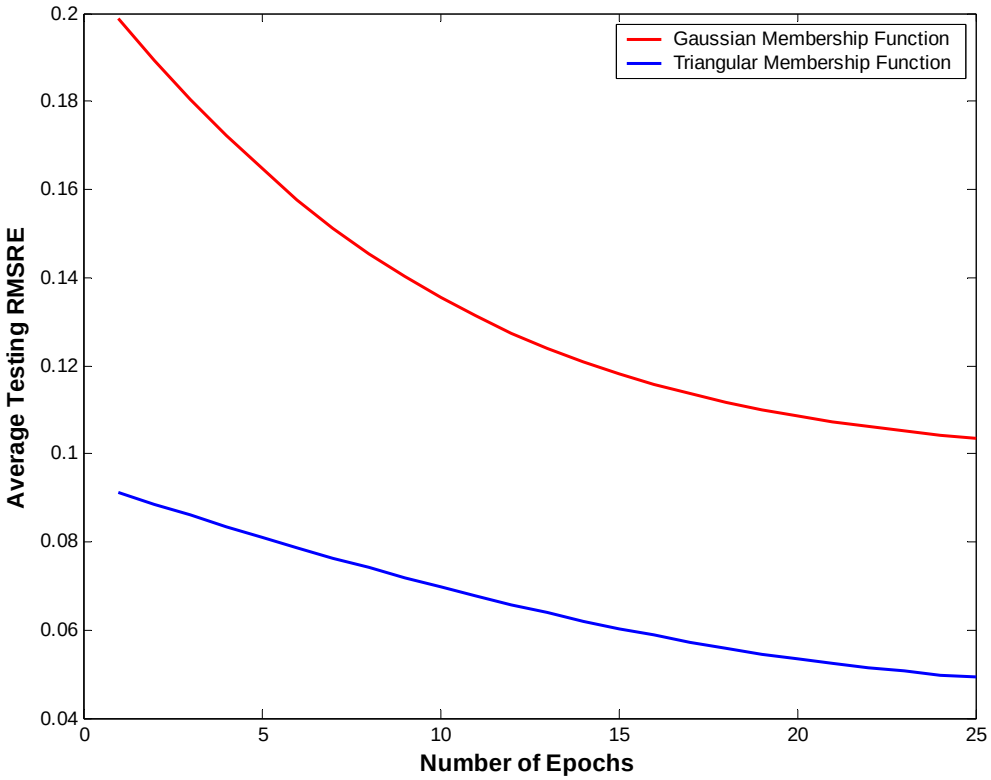

Figure 7-32: Average RMSRE graph of testing FLSs with Gaussian and triangular membership functions when size interval is [0, 100] KDSI.

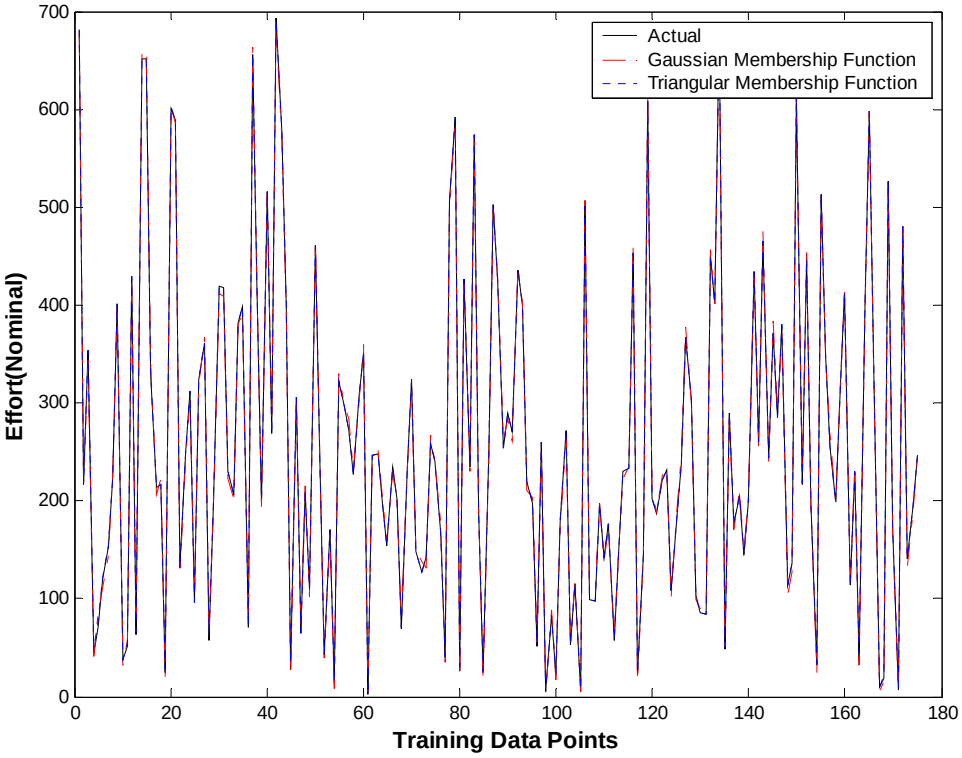

Figure 7-33: Prediction of nominal effort using trained FLSs with Gaussian and triangular membership functions on training dataset.

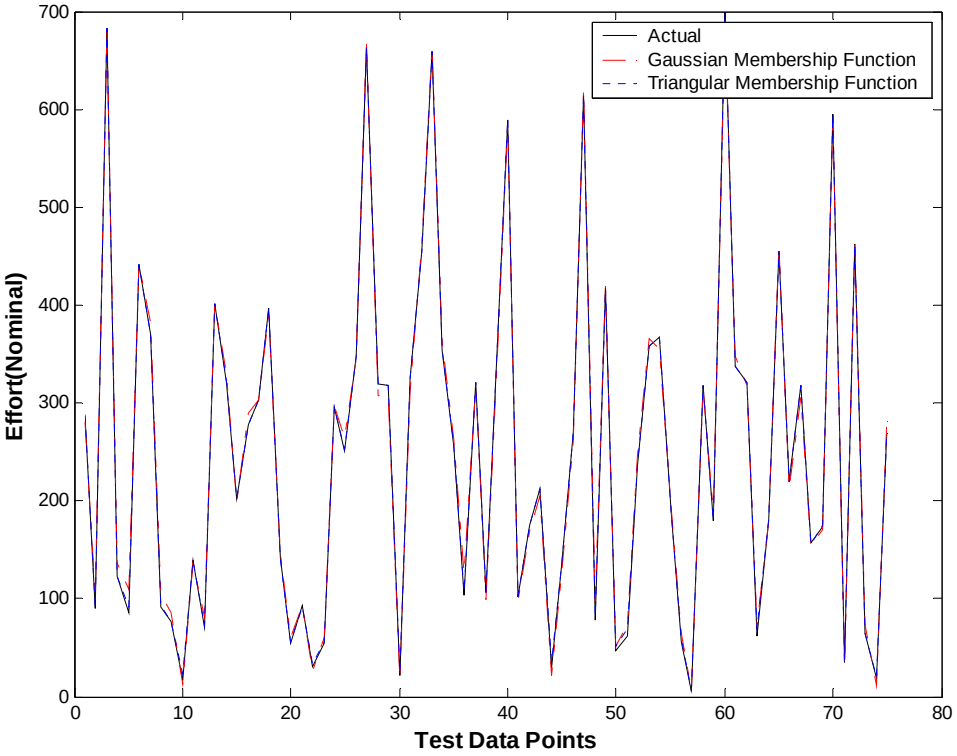

Figure 7-34: Prediction of nominal effort using trained FLSs with Gaussian and triangular membership functions on test dataset.

Table 7-11: Summary of Prediction Quality using Gaussian and Triangular Membership Functions on five different datasets, showing PRED(10) and PRED(25) on training and testing datasets. Interval of size is [0, 200] KDSI.

Experiment No.	Gaussian Membership Function				Triangular Membership Function			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	0.9950	1.0	0.9700	0.9500	1.0	1.0	0.9850	0.9900
2	0.9450	0.9200	0.8750	0.8800	0.9950	0.9600	0.9850	0.9400
3	0.9500	0.8800	0.8400	0.8200	0.9900	0.9600	0.9400	0.9200
4	0.9950	1.0	0.9600	0.9400	1.0	1.0	0.9900	0.9600
5	0.9950	0.9900	0.9800	0.9500	1.0	1.0	1.0	0.9900

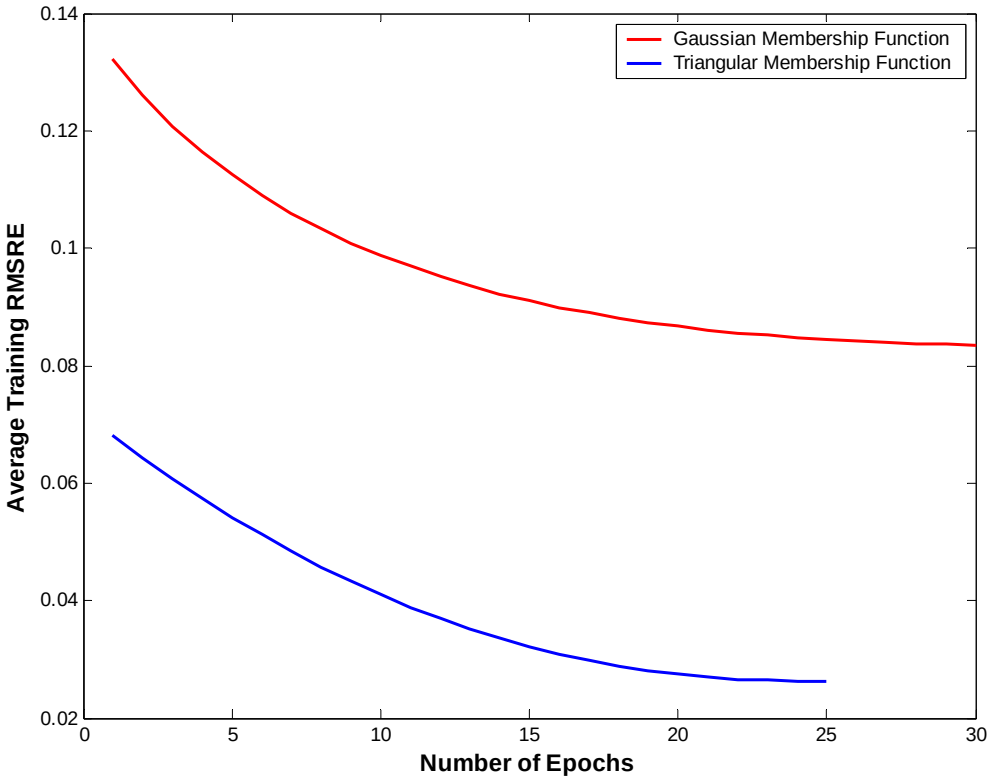

Figure 7-35: Average RMSRE graph of training FLSs with Gaussian and triangular membership functions when size interval is [0, 200] KDSI.

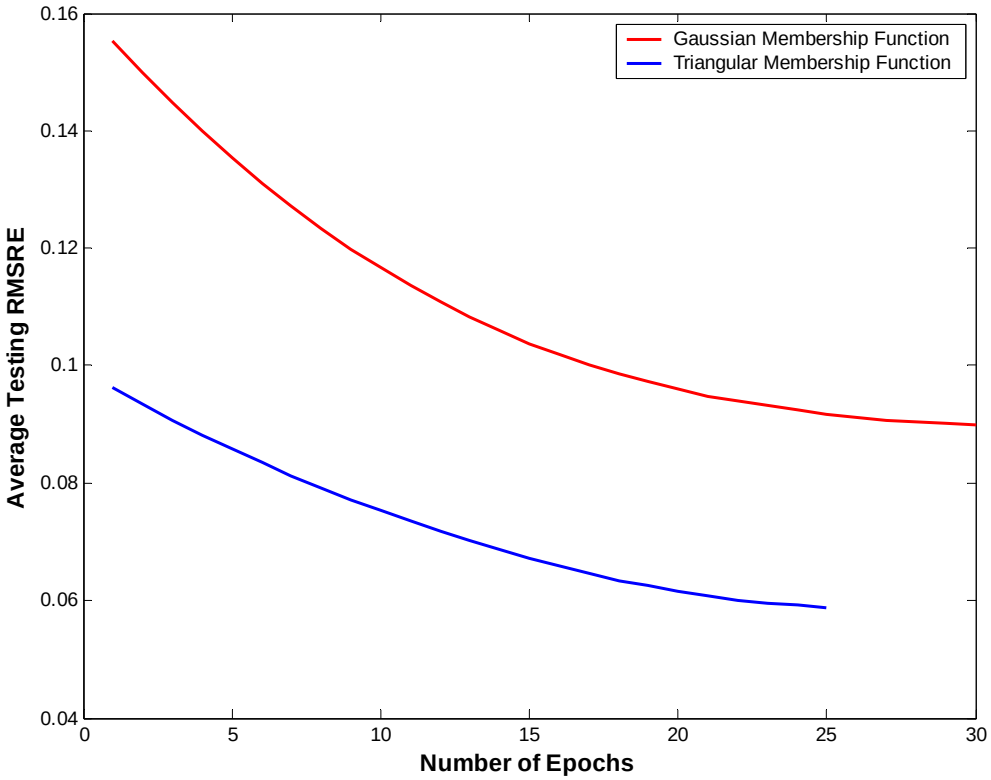

Figure 7-36: Average RMSRE graph of testing FLSs with Gaussian and triangular membership function when size interval is [0, 200] KDSI.

7.6. Experiment 5: Effect of Training Algorithms on FLS Based Effort Prediction Framework

This experiment is concerned with the comparison of steepest descent approach against heuristic based approach for training effort prediction system. In the following subsections, we will discuss algorithm for artificial dataset generation, training and testing of the prediction system, and the results.

7.6.1. Algorithm for Artificial Dataset Generation

In this experiment we have used Algorithm 7-4 for data generation because it allows generating the required datasets that contain mode and size values together with corresponding effort values.

7.6.2. Training and Testing

We have conducted 5 experiments using 5 different datasets. Each dataset consists of 250 data points and divided into 175 training and 75 testing data points.

7.6.3. Results and Discussion

It is evident from Table 7-12 and RMSRE graphs that the performance of training algorithm with steepest descent approach is better than heuristic based approach. We believe the reason for this is that steepest descent approach allows training each rule separately. Each rule gets its own copy of membership functions and the modification to the membership functions in a rule is applied locally. This is in contrast to the heuristic-

based algorithm where membership functions are modified rather than rules. If a modification due to any rule is needed, it is applied to the same copy of membership functions. Thus steepest descent training converges better than heuristic based approach and provides better results during activation or testing.

Table 7-12: Summary of Prediction Quality using Steepest Descent and Heuristic based approaches on five different datasets, showing PRED(10) and PRED(25) on training and testing datasets. Interval of size is [0, 100] KDSI.

Experiment No.	Steepest Descent				Heuristic			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	1.0	1.0	0.9486	0.9467	0.8343	0.7867	0.7714	0.6800
2	1.0	0.9867	0.9429	0.9067	0.8229	0.7467	0.7200	0.6735
3	0.9886	0.9867	0.9771	0.9733	0.8343	0.7867	0.7714	0.6800
4	1.0	1.0	0.9429	0.8800	0.7771	0.7733	0.6686	0.6800
5	1.0	1.0	0.9543	0.9333	0.7943	0.8533	0.7200	0.7467

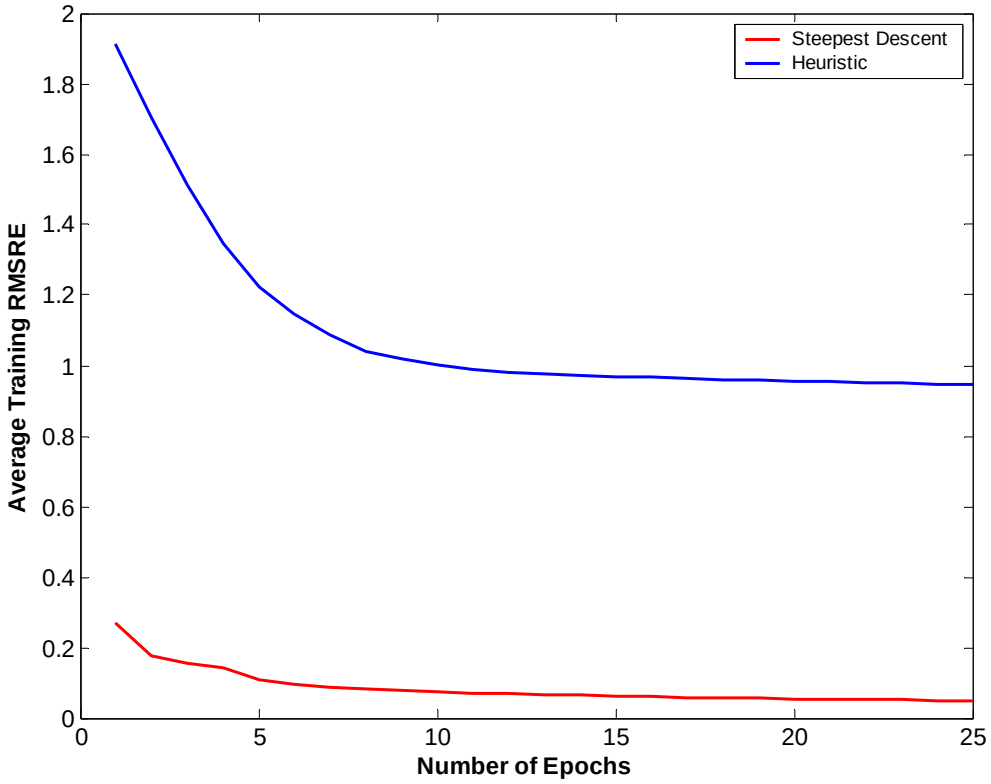

Figure 7-37: Average RMSRE graph of training FLSs using steepest descent and heuristic based approaches when size interval is [0, 100] KDSI.

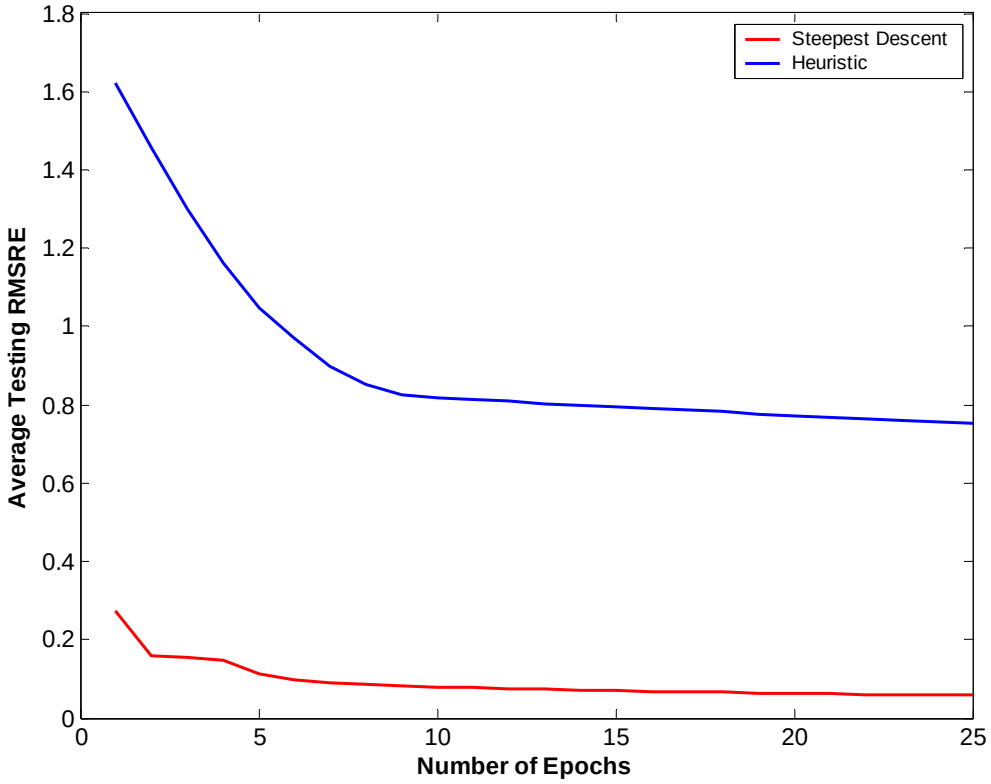

Figure 7-38: Average RMSRE graph of testing FLSs using steepest descent and heuristic based approaches when size interval is [0, 100] KDSI.

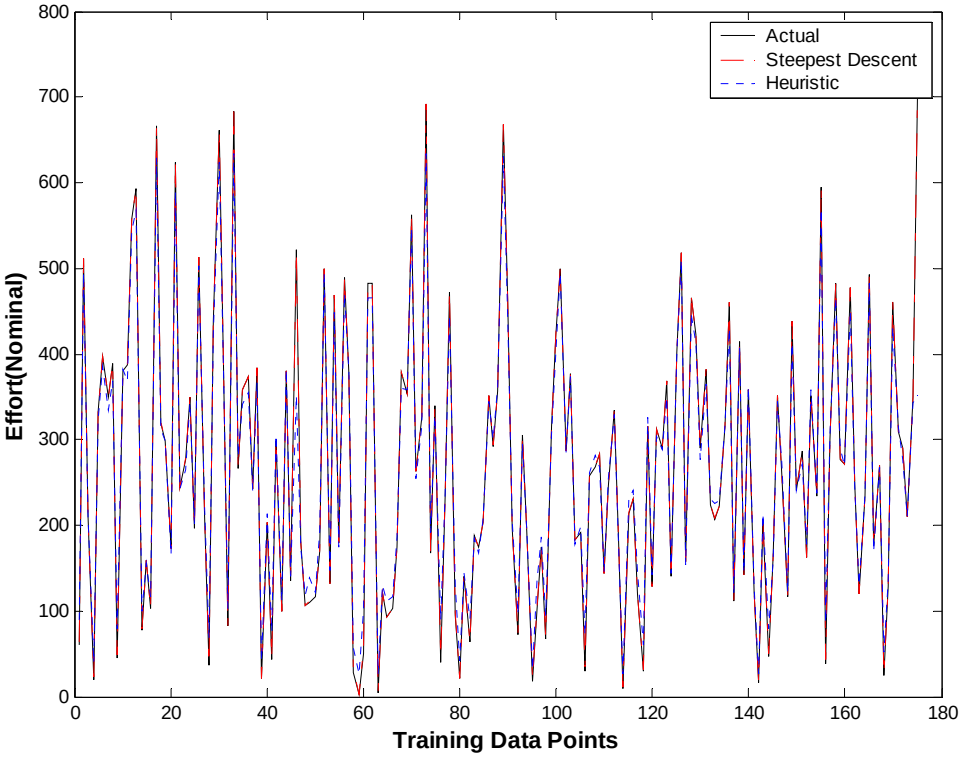

Figure 7-39: Prediction of effort using trained FLSs with steepest descent and heuristic based approaches on training dataset.

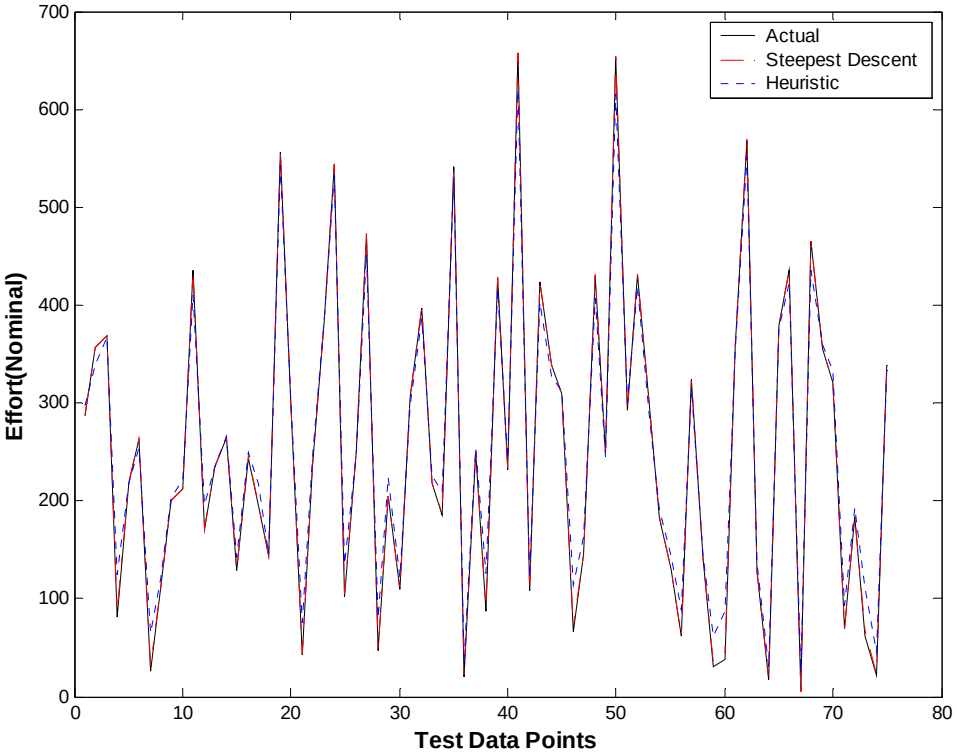

Figure 7-40: Prediction of effort using trained FLSs with steepest descent and heuristic based approaches on test dataset.

7.7. Experiment 6: Effect of Architecture on FLS Based Effort Prediction Framework

In this experiment, we have compared the two-antecedent effort prediction system against the three-antecedent effort prediction system. In the following subsections, we will discuss algorithm for artificial dataset generation, training and testing of the prediction systems, and the results.

7.7.1. Algorithm for Artificial Dataset Generation

In this experiment we have used Algorithm 7-4 for data generation with a little addition that we have further generated values of 17 cost drivers for each data point using Algorithm 7-3. In this case we have only applied first two steps of Algorithm 7-3 because laziness/ignorance issues are not need to be considered in this experiment.

7.7.2. Training and Testing

We have conducted 5 experiments using 5 different datasets. Each dataset consists of 200 data points and divided into 150 training and 50 testing data points.

7.7.3. Results and Discussion

It is evident from Table 7-9 and RMSRE graphs, see Figure 7-41 and 7-42, that the performance of architecture with two antecedents is better than one with three antecedents. This is because using two-antecedent architecture results in a simpler network than using three-antecedent. This simpler network maintains input-output

relationships much better than a complex one. In addition, one can notice the fluctuating behavior of RMSRE curves in Figure 7-41 and 7-42. This happens sometimes if the best parameter values through out the training iterations are not kept. Since in our experiments, we have also not kept the best parameters all the time, therefore sometimes parameters update caused increase in RMSRE which resulted in fluctuating behavior.

Table 7-13: Summary of Prediction Quality of two architectures, one with two antecedents and the other with three antecedents, on five different datasets, showing PRED(10) and PRED(25) on training and testing datasets.

Experiment No.	Mode and Size as Antecedents				Mode, Size and Effort Multipliers as Antecedents			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	0.9800	0.9800	0.9600	0.9800	0.9133	0.9200	0.6667	0.6200
2	0.9533	0.9600	0.8800	0.7400	0.9533	0.8600	0.8133	0.6000
3	0.8600	0.9200	0.6800	0.8000	0.9467	0.9200	0.7867	0.6400
4	0.9667	0.9400	0.8867	0.8200	0.9000	0.8400	0.7067	0.6600
5	0.9800	0.8800	0.9400	0.7600	0.9733	0.8600	0.7867	0.5400

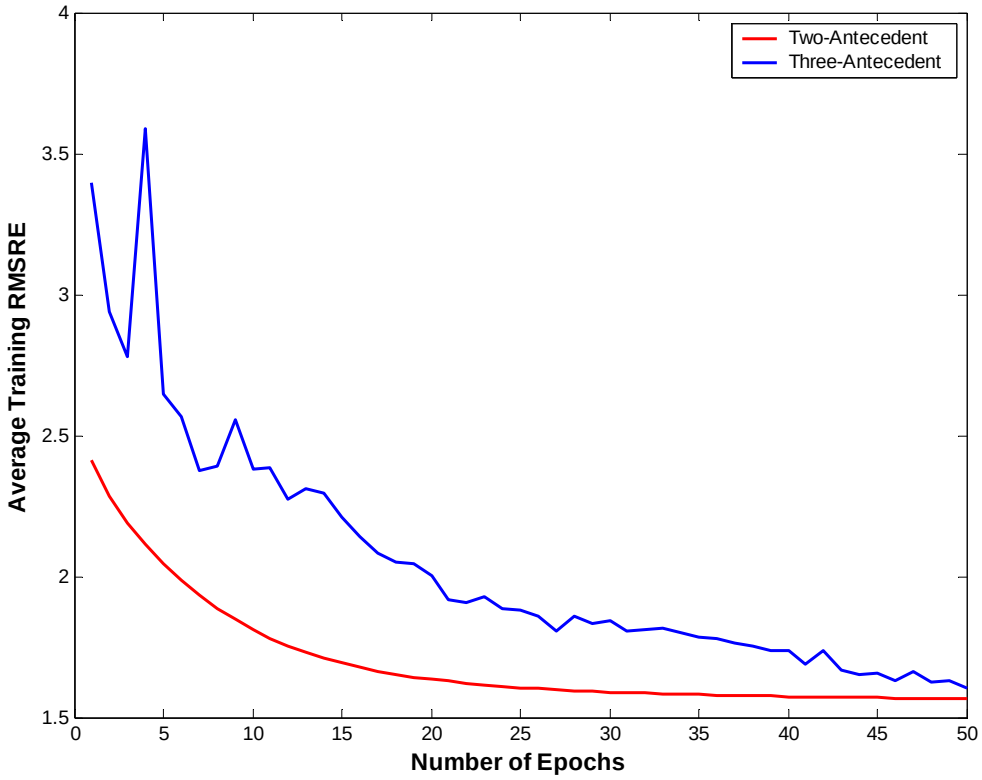

Figure 7-41: Average RMSRE graph of training FLSs with two and three antecedents.

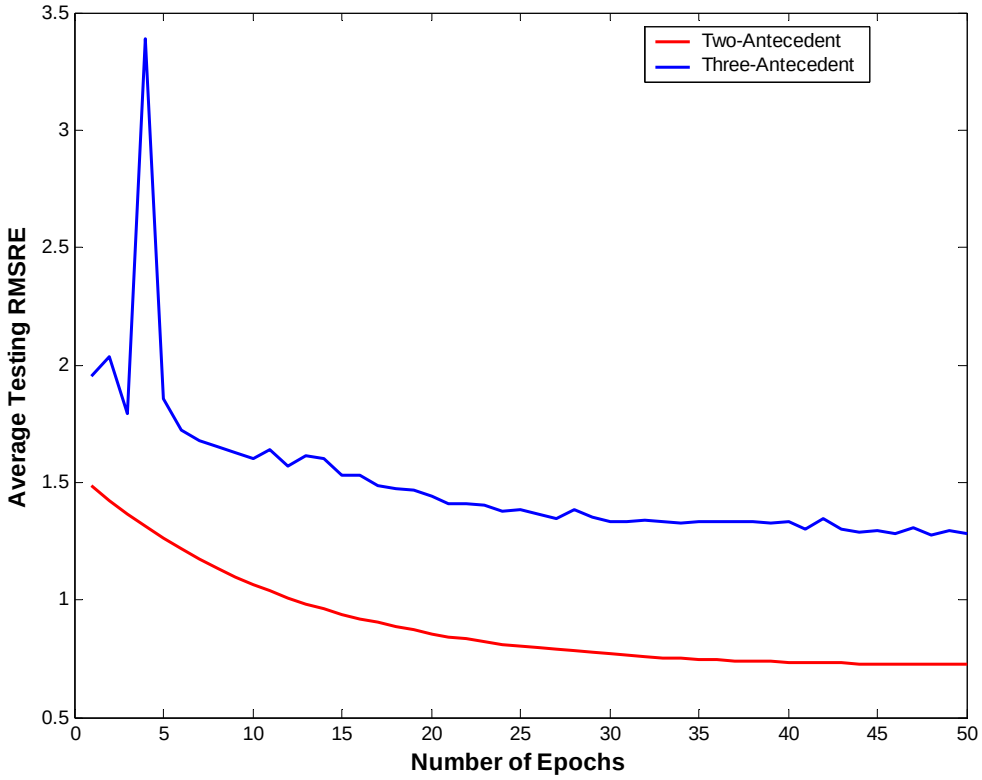

Figure 7-42: Average RMSRE graph of testing FLSs with two and three antecedents.

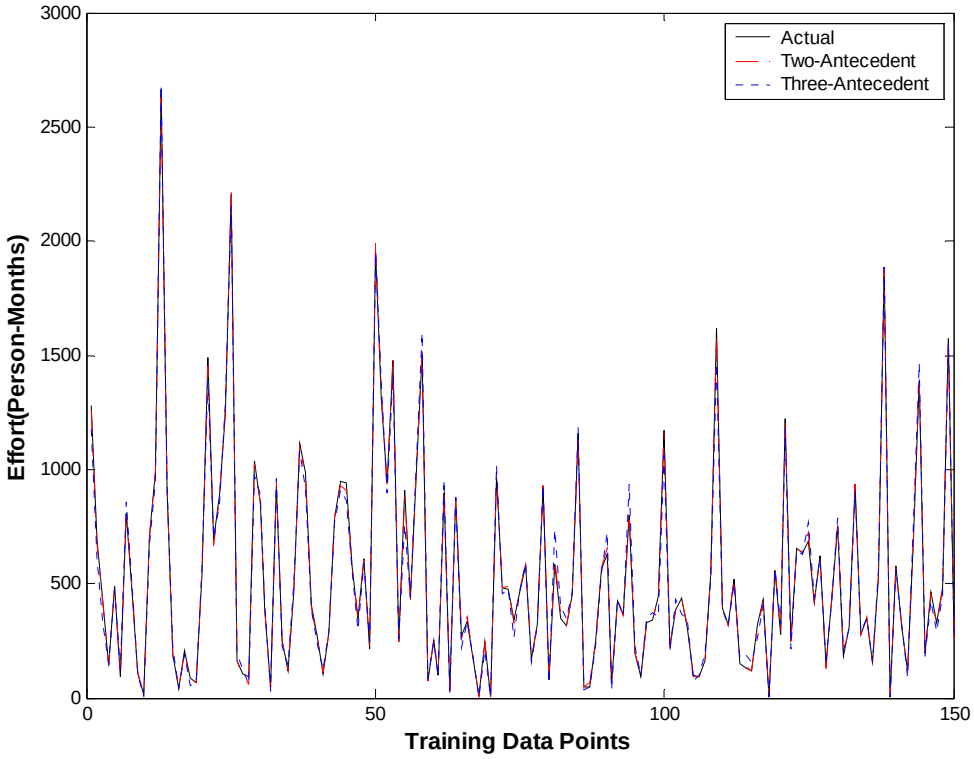

Figure 7-43: Prediction of effort (person-months) using trained FLSs with two and three antecedents on training dataset.

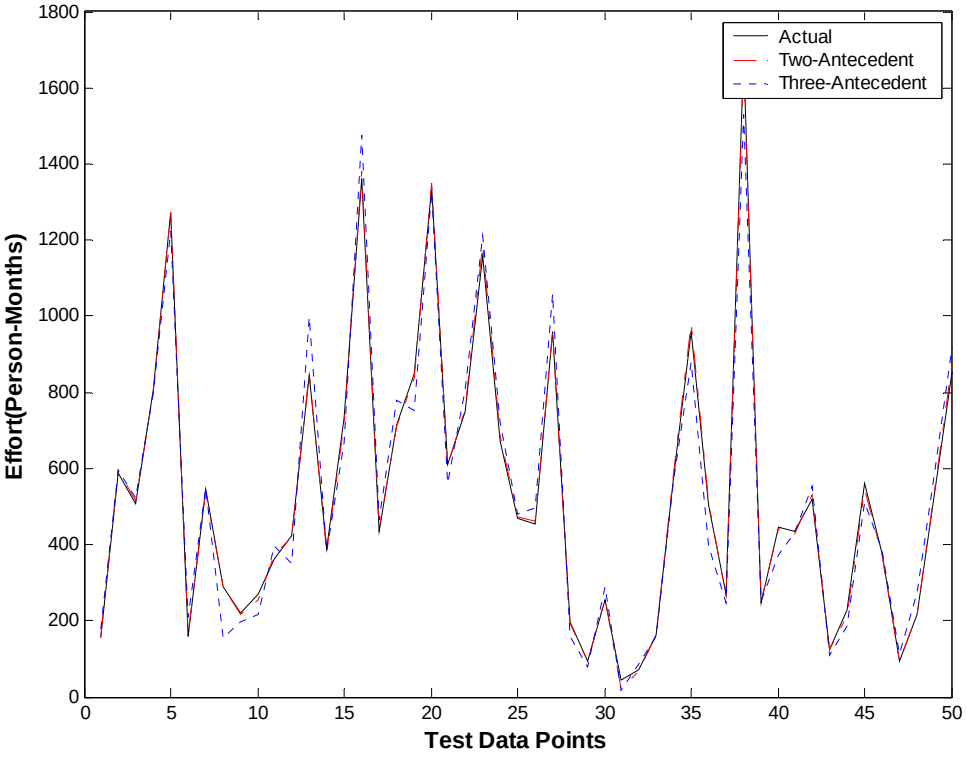

Figure 7-44: Prediction of effort (person-months) using trained FLSs with two and three antecedents on test dataset.

7.8. Experiment 7: Comparing FLS Based Two-stage Framework against the Proposed Multistage Framework for Handling Laziness/Ignorance

In this experiment we have compared the type-1 FLS based framework proposed by Ahmed *et al.* [2] and the multistage framework discussed in Chapter 5 for handling laziness/ignorance. In order to be able to compare both the frameworks on the same ground, we have considered all the stages of multistage framework to be type-1. We have employed the algorithms for generating datasets as discussed earlier.

7.8.1. Training and Testing

We have conducted 5 experiments using 5 different datasets. Each dataset consists of 200 data points and divided into 150 training and 50 testing data points.

7.8.2. Results and Discussion

It can be seen from Table 7-14 and corresponding RMSRE graphs; see Figure 7-45 and 7-46, that the performance of FLS based multistage framework is better than two-stage framework. This is because multistage framework directly addresses those portions (EAF computation) of the framework where uncertainty is introduced and it tries to handle this by using adaptive FLS. Handling uncertainty at its origin results in a simpler rule base, this in turn results in better prediction. The computed EAF is then allowed to get multiplied with the nominal effort to produce the actual effort. The two-stage framework,

on the other hand, does not take any special measures to prevent uncertainty from being propagated into the actual effort computation process.

Table 7-14: Summary of Prediction Quality of two-stage and multistage framework, on five different datasets, showing PRED(10) and PRED(25) on training and testing datasets.

Experiment No.	Two-stage Framework				Multistage Framework			
	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data	Prediction Accuracy Pred(25) on Training Data	Prediction Accuracy Pred(25) on Test Data	Prediction Accuracy Pred(10) on Training Data	Prediction Accuracy Pred(10) on Test Data
1	0.2867	0.3800	0.1133	0.1600	0.4200	0.5000	0.1600	0.1800
2	0.4067	0.3800	0.1267	0.1000	0.5400	0.4600	0.2600	0.2000
3	0.3067	0.3800	0.1000	0.1800	0.5000	0.4800	0.1933	0.1600
4	0.2400	0.3600	0.1133	0.1600	0.4867	0.5000	0.2333	0.2500
5	0.3333	0.4000	0.1400	0.1600	0.5067	0.4200	0.1800	0.1600

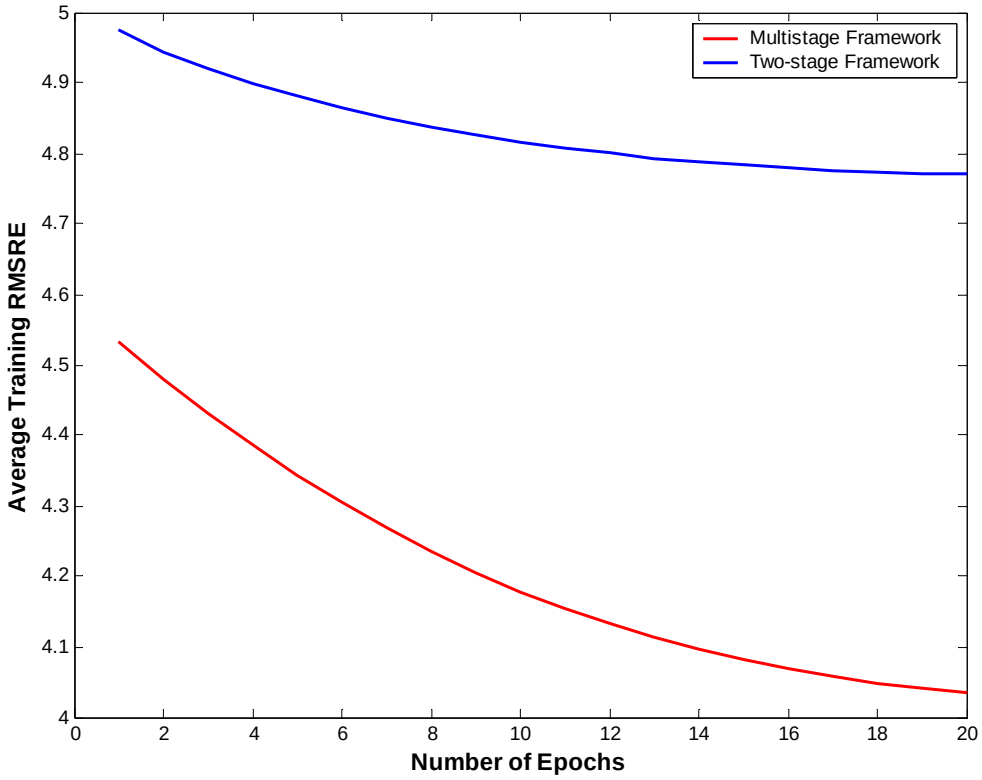

Figure 7-45: Average RMSRE graph of training two-stage and multistage frameworks.

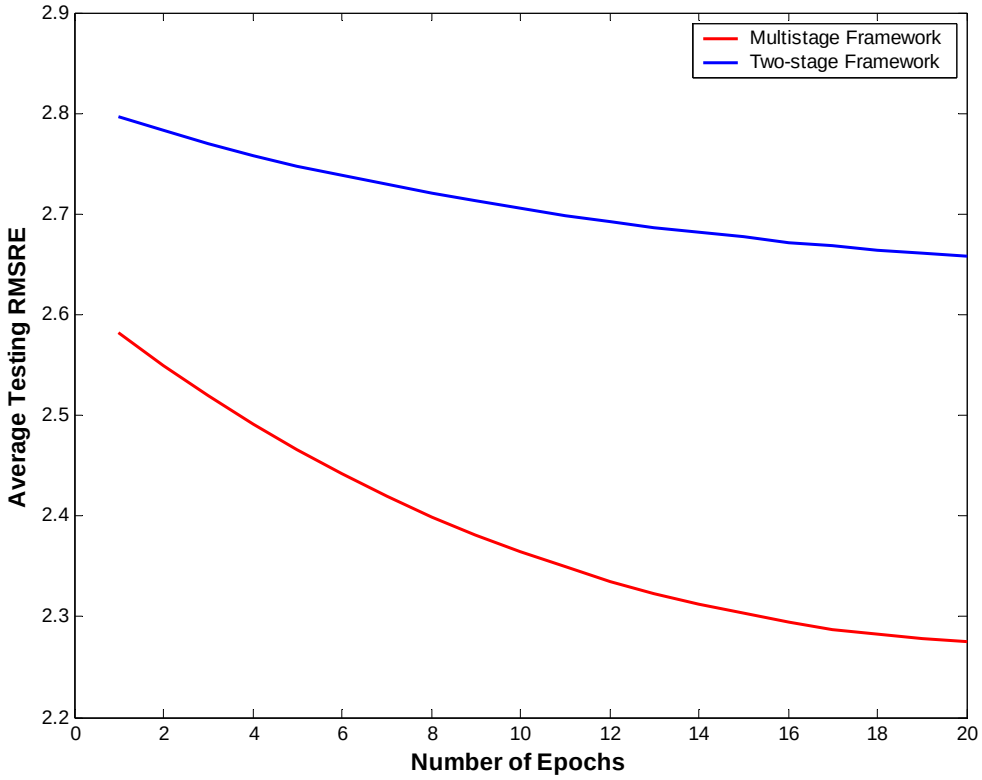

Figure 7-46: Average RMSRE graph of testing two-stage and multistage frameworks.

CHAPTER 8

CONCLUSION

8.1. Introduction

This chapter provides a summary of the contributions presented in this work. It also provides some directions for the future research.

8.2. Summary of Contributions

The thesis research has resulted in the following contributions to knowledge:

i. Some attributes that portray a picture of the credibility of software size metrics are identified. These attributes do not only assess the credibility of size metrics from user perspective but they also consider the internal details as to how a particular metric is designed, what information is captured, what level of details are induced into the computation process, etc. These attributes should be kept in mind while proposing new software size metrics.

ii. Guided by the presented attributes, an extensive critical survey of some of the existing software size metrics is presented. Besides other findings it was found out that existence of various metrics leads to uncertainty, which results in uncertainty when these metrics are related to effort. Moreover, even within a single metric there are sources of uncertainty e.g., subjective involvement. This situation led us to develop such framework that can handle imprecision and uncertainty within a single framework.

iii. Four types of FLS based effort prediction systems are defined. This classification is performed on the bases of the nature of inputs e.g., singletons vs. non-singleton; and whether systems are capable of handling various sources of uncertainty, refer to Chapter 1, in the inputs.

iv. Framework for handling imprecision and uncertainty using type-2 fuzzy logic system are presented. The uncertainty due to the existence of various size metrics, conflicting experts' opinion, ignorance and laziness is encountered. The experiments conducted revealed that type-2 FLS handles uncertainty better than type-1 FLS.

v. The impact of various FLS parameters on FLS based effort prediction is studied empirically. The parameters studied are defuzzification method (e.g., height vs. modified height), shape of membership functions (e.g., Gaussian vs. triangular) and propagated error (e.g., relative vs. normalized).

vi. The impact of the nature of training algorithm on the FLS based effort prediction is investigated empirically. Steepest descent was shown to be performing better than a heuristic based algorithm.

vii. The FLS based effort prediction framework that makes use of various components of COCOMO can be constructed in different ways. Some possible architectures are two-antecedent system and three-antecedent system. These systems were compared and it was revealed that the two-antecedent system showed better accuracy than the three-antecedent system.

viii. The type-1 FLS based framework proposed by Ahmed *et al* [2] and the multistage framework, discussed in Chapter 5, for handling laziness/ignorance were compared. It is revealed that prediction system constructed by employing multiple stage framework handles uncertainty better than the one constructed by using framework proposed by Ahmed et al.

8.3. Future Research

It is difficult to cover all the facets associated with the problem of effort prediction in the limited amount of time allocated to the thesis work. The same is also true with this work. Following are some directions for future research:

i. Developing formal metrics for the evaluation of size metrics based on our generic set of attributes is a potential candidate for further research.

ii. Looking at the prospects of adding some other attributes to the attributes set.

iii. Developing a rating scheme for software size metrics based on our generic set of attributes. In this context one must investigate how important a particular attribute is or to what extent a particular attribute contributes to the credibility of size metrics.

iv. Developing a new software size metric taking into consideration our proposed attributes set.

v. Type-2 FLS developed by Mendel still has a limitation that it only allows uniform distribution (interval sets) in the third dimension of the fuzzy sets. This means that contradicting experts' opinions can not be weighted differently. Some future work can be directed towards this to develop such FLS that can allow other distributions e.g., Gaussian and triangular.

vi. In order to perform experiments in this thesis, we tried to acquire some real datasets but we could not manage to get them in time. Therefore, the experiments were conducted on artificially generated datasets. Hence, a need to evaluate the prediction performance of the proposed framework on real datasets still persists. In this context, one can contact some organizations to acquire such datasets that can be used to evaluate the framework's performance.

BIBLIOGRAPHY

[1] Adam A., Jaralla, A., and Ahmed, M.: "Can Cohesion Predict Fault Density?", The 4th ACS/IEEE International Conference on Computer Systems and Applications, Dubai/Sharjah, UAE, March 8-11, 2006.

[2] Ahmed, M., A., Saliu, M., O. and Al-Ghamdi, J.: "Adaptive Fuzzy Logic Based Framework for Software Development Effort Prediction", Information and Software Technology, Vol. 47, No. 1, January, 2005, pp. 31-48.

[3] Albrecht, A., J.: "Measuring Application Development Productivity", In Proceedings of the IBM Applications Development Symposium, SHARE-Guide, 1979, pp. 83-92.

[4] Angelis, L., Stamelos, I. and Morisio, M.: "Building a Software Cost Estimation Model Based on Categorical Data", The 7th IEEE International Software Metric Symposium, London, England, 2001.

[5] Antoniol, G., Lokan, C., Caldiera, G. and Fiutem, R.: "A Function Point-Like Measure for Object Oriented Software" Empirical Software Engineering, 4, 1999, pp. 263-287.

[6] Bibi, S., Stamelos, I., and Aggelis, L.: "Bayesian Belief Networks as a Software Productivity Estimation Tool", 1st Balkan Conference in Informatics, Thessaloniki, Greece, November 2003.

[7] Boehm, B., Abts, C., and Chulani, S.: "Software Development Cost Estimation Approaches: A Survey", University of Southern California Centre for Software Engineering, Technical Report, USC-CSE-2000-505, 2000.

[8] Boehm, B., Chulani, S., and Reifer, D.: "The Rosetta Stone: Making COCOMO 81 Files Work With COCOMO II", 1998.

[9] Boehm, B., Clark, B., Horowitz, E., Madachy, R., Shelby, R., and Westland C.: "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0", Annals of Software Engineering, 1995.

[10] Boehm, B. W.: "Software Engineering Economics", Englewood Cliffs, NJ, Prentice-Hall, 1981.

[11] Briand, L., C., Morasca, S. and Basili, V., R.: "Property Based Software Engineering Measurement", IEEE Transaction on Software Engineering, Vol. 22, No. 1, January, 1996, pp. 68-86.

[12] Briand, L., Melo, W. and Wust, J.: "Assessing the Applicability of Fault-Proneness Modelsa Across Object Oriented Software Projects", IEEE Transactions on Software Engineering 28, 7, July 2002.

[13] Carbone, M. and Santucci, G.: "Fast&&Serious: A UML Based Metric for Effort Estimation", 6th ECOOP Workshop on Quantitative Approaches in Object Oriented Software Engineering, June 11, 2002.

[14] Center for Software Engineering, http://sunset.usc.edu/research/COCOMOII/cocomo_main.html

[15] Chen, Y., Boehm, B., W., Madachy, R., and Valerdi, R..: "An Empirical Study of eServices Product UML Sizing Metrics", In Proceedings of the ACM-IEEE

International Symposium on Empirical Software Engineering (ISESE 2004), IEEE-CS Order No. P2165, Redondo Beach CA, USA, August 19-20, 2004, pp. 199-206.

[16] Chidamber, S. R. & Kemerer C. F.: "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, Vol. 20, No. 6, June, 1994, pp: 476-493.

[17] Clark, B., Chulani, S. and Boehm, B.: "Calibrating the COCOMO II Post-Architecture Model ", Proceedings of International Conference on Software Engineering, 1998, pp. 477-480.

[18] Conte, S., Dunsmore, H. and Shen, V.: "Software Engineering Metrics and Models", Benjamin-Cummings, Menlo Park, CA, 1986.

[19] Costagliola, G., Ferrucci, F., Tortora, G. and Vitiello, G.: "Class Point: An Approach for the Size Estimation of Object-Oriented Systems", IEEE Transactions on Software Engineering, Volume 31, Issue 1, January, 2005, pp. 52-74.

[20] Fenton, N., E., and Pfleeger, S., L.: "Software Metrics: A rigorous and Practical Approach", Second Edition, PWS Publishing Company, 1997.

[21] "Full Function Points: Counting Practices Manual", Technical Report 1997-04, Montreal, Software Engineering Management Research Laboratory and Software Engineering Laboratory in Applied Metrics, University of Quebec Montreal, Canada, 1997.

[22] Hareton, L., and Zhang F.: "Software Cost Estimation", Department of Computing, Hong Kong Polytechnic University. http://paginaspersonales.deusto.es/cortazar/doctorado/articulos/leung-andbook.pdf

[23] Hastings, T., E. and Sajeev, A., S., M.: "A Vector-Based Approach to Software Size Measurement and Effort Estimation", IEEE Transactions on Software Engineering, Volume 27 , Issue 4, April 2001, pp. 337-350.

[24] Hastings, T., E., and Sajeev, A., S., M.: "A Vector-Based Approach to Software Size Measurement and Effort Estimation", IEEE Transactions on Software Engineering, Vol. 27, No. 4, April 2001.

[25] Hastings, T., E.: "Adapting Function Points to Contemporary Software Systems: A Review of Proposals", Proceedings of the 2nd Australian Conference on Software Metrics (ACOSM '95), 1995, pp. 103-114.

[26] Hodgkinson, A.C. and Garratt, P. W.: "A NeuroFuzzy cost estimator", In Proceedings of the 3rd International Conference on Software Engineering and Applications - SAE, 1999, pp. 401-406.

[27] Idri, A. and Abran, A.: "COCOMO Cost Model Using Fuzzy Logic", The 7th International Conference on Fuzzy Theory and Technology, Atlantic City, New Jersy, March 2000.

[28] International Software Benchmarking Standards Group (ISBSG), http://www.isbsg.org/

[29] Janaki, R., D. and Raju, S., V., G., K.: "Object Oriented Design Function Points", In Proceedings of the 1st Asia-Pacific Conference on Quality Software (APAQS'00), 30-31 October, 2000, pp. 121-126.

[30] Jeffery, R. and Low, G.: "Function Points and Their Use", Australian Computer J., Vol. 29, No. 4, 1997, pp. 148-156.

[31] Jenson, R. L. & Bartley, J. W.: "Parametric Estimation of Programming Effort: An Object-Oriented Model", Journal of Systems and Software, Vol. 15, 1991, pp. 107-114.

[32] Jones, C.: "Applied Software Measurement", New York, N.Y.: McGraw-Hill, 1991.

[33] Jorgensen, M., and Molokken, K.: "Combination of software development effort prediction intervals: Why, when and how?", In Proceedings of the Fourteenth IEEE Conference on Software Engineering and Knowledge Engineering (SEKE'02), Ischia, Italy, 2002, pp. 425-428.

[34] Karner, G.: "Resource Estimation for Objectory Projects", Objectory Systems, 1993.

[35] Kathleen Peters: "Software Project Estimation", Software Productivity Center Inc., 1999.

[36] Kauffman, R. and Kumar, R.: "Modeling Estimation Expertise in Object Based CASE Environments", Stern School of Business Report, New York University, January, 1993.

[37] Kirsopp, C., Shepperd, M., J. and Hart, J.: "Search Heuristics, Case-Based Reasoning and Software Project Effort Prediction", Genetic and Evolutionary Computing Conference (GECCO 2002), New York, AAAI, 2002.

[38] Kirsopp, C. and Shepperd M., J.: "Making Inferences with Small Number of Training Sets", The 6th International Conference on Empirical Assessment and

Evaluation in Software Engineering, Keele University, Staffordshire, UK, April 8th -10th, 2002.

[39] Kirsten Ribu: "Estimating Object-Oriented Software Projects with Use Cases", Master of Science Thesis, Department of Informatics, University of Oslo, Oslo, Norway, November 7, 2001.

[40] Kitchenham, B., Pfleeger, S., L. and Fenton, N.: "Towards a Framework for Software Measurement Validation", IEEE Transaction on Software Engineering, Vol. 21, No. 12, 1995, pp. 929-944.

[41] Klir, G., J. and Wierman, M., J.: "Uncertainty-Based Information", Heidelberg, Germany: Physica-Verlag, 1998.

[42] Kusumoto, S., Matukawa, F., Inoue, K., Hanabusa, S. and Maegawa, Y.: "Estimating Effort by Use Case Points: Method, Tool and Case Study", In Proceedings of the 10th International Symposium on Software Metrics, (METRICS'04), Volume 00, September, 2004.

[43] Laranjeira, L.: "Software Size Estimation of Object-Oriented Systems", IEEE Transactions on Software Engineering, Vol. 16, No. 5, May, 1990, pp: 510 – 522.

[44] Larsen, P., M.: "Industrial Applications of Fuzzy Logic Control", International Journal of Man-Machine Studies, Vol. 12, No. 1, 1980, pp. 3-10.

[45] Lockheed Martin: "Advanced Concept Center training materials", Object Oriented Size and Cost Estimation, 1994.

[46] MacDonell, S., G., and Gray A., R.: "A Comparison of Modeling Techniques for Software Development Effort Prediction", In Proceedings of the 1997

International Conference on Neural Information Processing and Intelligent Information Systems, Denedin, Newzealand, Springer-Verlag (1997), 869-872.

[47] Mamdani, E., H.: "Applications of Fuzzy Algorithms for Simple Dynamic Plant", Proc. IEEE 121, 12 (1974).

[48] Mendel, J., M. and Liang, Q.: "Pictorial Comparison of Type-1 and Type-2 Fuzzy Logic Systems", In Proceedings of IASTED International Conference on Intelligent Systems & Control, Santa Barbara, CA, Oct., 1999.

[49] Mendel, J., M.: "Uncertain Rule-Based Fuzzy Logic Systems", Prentice-Hall, Upper Saddle River, NJ 07458, 2001.

[50] Minkiewicz, A. F.: "Objective Measures", Software Development, June 1997, pp: 43-47.

[51] Musilek, P., Pedryez, W., Succi, G. and Reformat, M.: "Software Cost Estimation with Fuzzy Models", Applied Computing Review, Vol. 8, No. 2, pp. 24-29, 2000.

[52] NASA JSC: Basic COCOMO Software Cost Model, http://www.jsc.nasa.gov/bu2/COCOMO.html.

[53] Nauck, D.: "Data Analysis with Neuro-Fuzzy Methods", Habilitation Thesis, University of Magdeburg, 2000.

[54] Pittman, M.: "Lessons Learned in Managing Object-Oriented Development", IEEE Software, January, 1993.

[55] Putnam, L. H.: "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", IEEE Transactions on Software Engineering, Vol. 4., No. 4, pp. 345-361, 1978.

[56] Rahman, Q. A.: "Dealing with Imprecision and Uncertainty while Developing Software Quality Models", Masters Thesis, Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, 2005.

[57] Royce, W.: "Software Project Management: A Unified Framework", Addison Wesley, 1998.

[58] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W.: "Object-Oriented Modeling and Design", Prentice-Hall, Rational Software Corporation, 1997b, Unified Modeling Language, Version 1.1, 1991.

[59] Russell, Geddes and Grossett: "Webster's New Dictionary and Thesaurus", Geddes and Grosset Ltd., New Lanark, Scotland, 1990.

[60] Russell, S. and Norvig, P.: "Artificial Intelligence: A Modern Approach", 1st Edition, Prentice-Hall Inc., 1995.

[61] Saliu, M., Ahmed, M. and AlGhamdi, J.: "Towards Adaptive Soft Computing Based Software Effort Prediction", In IEEE Meeting of the Fuzzy Information Processing NAFIPS, Volume 1, 27-30 June, 2004, pp. 16-21.

[62] Saliu, M.O., and Ahmed, M.A., "Soft Computing Based Effort Prediction Systems – A Survey", A Chapter in E. Damiani, L. C. Jain, and M. Madravio (EDs), Soft Computing in Software Engineering, Springer-Verlag Publisher, July 2004, ISBN 3-540-22030-5.

[63] Saliu, M.: "Adaptive Fuzzy Logic Based Framework for Software Development Effort Prediction", Master Thesis, Information and Computer Science

Department, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, 2003.

[64] Schofield C.: "Non Algorithmic Effort Estimation Techniques". Technical Report, Department of Computing, Bournemouth University, England, TR98-01, March 1998.

[65] Smith, J.: "The Estimation of Effort Based on Use Case", IBM Rational Software, White Paper, 1999.

[66] "Software estimation, benchmarking, productivity, risk analysis, and cost information for software developers and business", http://www.isbsg.org/

[67] Srinivasan, K., and Fisher, D.: "Machine Learning Approaches to Estimating Software Development Effort", IEEE Transactions on Software Engineering, Vol. 21, No. 2, 1995.

[68] Strike, K., El-Emam, K. and Madhavji M.: "Software Cost Estimation with Incomplete Data", IEEE Transactions on Software Engineering, Vol. 27, No. 10, Oct. 2001.

[69] Stutzke, R., D.: "Software Estimation Technology: A Survey", IEEE Software Engineering Project Management, 1997.

[70] Symons, C., R.: "Function Point Analysis: Difficulties and Improvements", IEEE Transaction on. Software Engineering, Vol. 14, No. 1, 1988, pp. 2-11.

[71] Symons, C., R.: "Software Sizing and Estimating: Mk II FPA", Chichester, England, John Wiley, 1991.

[72] Teologlou, G.: "Measuring Object Oriented Software with Predictive Object Points", In 10th Conference on European Software Control and Metrics, May 1999.

[73] Liang, T. and Noore, A.: "Multistage Software Estimation", Proceedings of the 35th Southeastern Symposium on System Theory, 16-18 March, 2003, pp. 232 – 236.

[74] Uemura, T., Kusumoto, S. and Katsuro, I.: "Function Point Measurement Tool. for UML Design Specification", In Proceedings of the 6th International Symposium on Software Metrics, November 4-6, 1999.

[75] Walston, C., E., and Felix C., P.: "A Method of Programming Measurement and Estimation", IBM Systems Journal, vol. 16, no. 1, 1977, pp. 54-73.

[76] Wang, L.: "Adaptive Fuzzy System and Control: Desing and Stability Analysis", Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1994.

[77] Warmer, J. and Objecten, K.: "The Future of UML", http://www.klasse.nl/english/uml/uml2.pdf.

[78] Whitmire, S., A.: "3D Function Points: Scientific and Real-Time Extensions to Function Points", In the Proceedings of. Pacific Northwest Software Quality Conference, 1992.

[79] Wu, L.: "The comparison of the Software Cost Estimating Methods", University of Calgary, Calgary, Canada, 1997, http://sern.ucalgary.ca/courses/seng/621/W97/wul/seng621_11.html.

[80] Zadeh, L., A.: "Fuzzy Sets", Information and Control 8, 1965, pp. 338-353.

[81] Zadeh, L., A.: "The concept of a Linguistic Variable and Its Application to Approximate Reasoning-1", Information Sciences, Volume. 8, 1975, pp. 199-249.

[82] Zimmermann., H.: "Fuzzy Set Theory and Its Applications", Kluwer Academic Publishers, Third Edition, 1996.

[83] Zonglian, F. and Xihui, L.: "f-COCOMO: Fuzzy Constructive Cost Model in Software Engineering", Proceedings of IEEE International Conference on Fuzzy Systems (IEEE 1992), pp. 331-337.

VITA

Syed Zeeshan Muzaffar, who hails from Karachi, Pakistan, received Bachelor of Engineering degree in Computer and Systems Engineering from NED University of Engineering & Technology, Karachi, Pakistan in April 2001. He served in the capacity of a lecturer at Computer and Information Systems Engineering Department in NED University of Engineering & Technology, in the year 2001. He worked as a Software Quality Assurance Engineer in Precience Technologies (PVT) Ltd., in the year 2002. He joined KFUPM as Research Assistant in February 2003. He is the author of 4 research publications in various conferences.